VIRGINIA COMMONWEALTH UNIVERSITY

Virginia Commonwealth University
VCU Scholars Compass

Theses and Dissertations | Graduate School

2011

Development of a Multiple Vehicle Collaborative Unmanned Aerial System

Lloyd B. Mize IV
Virginia Commonwealth University

Follow this and additional works at: http://scholarscompass.vcu.edu/etd

 Part of the Engineering Commons

© The Author

Downloaded from
http://scholarscompass.vcu.edu/etd/2527

This Thesis is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.

Development of a Multiple Vehicle Collaborative Unmanned Aerial System

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University

by

Lloyd B. Mize IV

Director: Dr. Robert H. Klenke
Associate Professor of Electrical and Computer Engineering

Virginia Commonwealth University
Richmond, Virginia
August 2011

Acknowledgements

I would like to thank all the people that helped me in progress and completion of this work. Specifically I would like to thank my advisor, Dr. Klenke, and my committee members, Dr. McCollum and Dr. Primeaux, for their help and support. I would also like to thank the faculty of the Engineering Department and their help throughout my years at VCU. As well, I would like to thank my fellow graduate students, Tim Bakker, Robert Demott, and Jose Ortiz, for their aid and friendship. I would also like to thank my family for their love and support. Above all, I would like to thank the Lord Jesus Christ for His strength and mercy during this time and throughout my life.

Table of Contents

List of Tables vi

List of Figures vii

List of Abbreviations x

Abstract xii

Chapter 1: Introduction 1

1.1 Overview 1

1.2 Motivation and Goals 1

1.3 Research Scope 3

1.4 Thesis Layout 3

Chapter 2: Background 5

2.1 Collaborative System Review 5

2.1.1 University of California, Berkeley 5

2.1.2 Massachusetts Institute of Technology 8

2.1.3 Brigham Young University 10

2.2 Path Planning Review 12

2.2.1 Raster Search Pattern 12

2.2.2 Entropy Based Search Pattern 14

2.2.3 Voronoi Graph 16

2.2.4 Robust Real-Time Route Planning 17

2.3 VCU System Overview 19

2.3.1 MiniFCS 20

2.3.2 VCU Aerial Communications Standard....22
Chapter 3: Networking, API, and Fragmentation....24
3.1 Digi API Mode....24
3.2 Packet Structure....25
3.3 Data Transmission and Reception....30
3.4 Arbitration Schemes....34
Chapter 4: GCS and Interface Updates....38
Chapter 5: Mission Control System Hardware....43
5.1 MCS Hardware Description....43
5.2 MCS Physical Layout....45
Chapter 6: Mission Control System Software....49
6.1 MCS Main Loop....49
6.2 Collaborative Modes....51
6.3 Search Area FSM....52
6.3.1 Reset....53
6.3.2 Start....53
6.3.3 Ping....54
6.3.4 Confirm....54
6.3.5 Segment....54
6.3.6 Vote and Pass....56
6.3.7 Verify....57
6.3.8 Path....57
6.3.9 Fly and Loiter....65

6.4 MCS Status Messaging 67

Chapter 7: Results and Flight Testing.....69

7.1 Collaborative Vote Validation 71

7.2 Speed Comparison 73

7.3 Wind Simulations.....75

7.4 Flight Testing 78

Chapter 8: Conclusions and Future Work.....83

8.1 Conclusions.....83

8.2 Future Work 84

List of Tables

Table 2.1: Layout of VACS Packet 23

Table 3.1: API Frame Types and Byte-Value 26

Table 3.2: Example for Transmit Request Packet Layout 27

Table 3.3: Frame Packet Structure 28

Table 3.4: Polling Mechanism Results 35

Table 3.5: Collaborative Operation Polling Results 36

Table 6.1: Status Packet Structure 67

Table 6.2: “Updates” Bit-Level Layout 68

Table 7.1: Plane Description for Voting Test 71

Table 7.2: Operating Altitudes in Speed Test 75

Table 7.3: Time Comparison for Speed Test 75

List of Figures

Figure 2.1: Guaranteed Search....6
Figure 2.2: Berkeley Sig Rascal....7
Figure 2.3: HILS setup for MIT....9
Figure 2.4: Tower Trainer....9
Figure 2.5: Kestrel Autopilot, Test Vehicles, and Ground Station....12
Figure 2.6: Raster Pattern....13
Figure 2.7: Entropy Graph....14
Figure 2.8: Path and Resulting Entropy Reduction....15
Figure 2.9: Final Path and Resulting Entropy Graph....16
Figure 2.10: Voronoi Graph of Radar Sites....17
Figure 2.11: Cost Map....18
Figure 2.12: Fan Out of Optimum Path....19
Figure 2.13: MiniFCS PCB Layout....20
Figure 2.14: Cross-track Error Vector Diagram....21
Figure 2.15: MiniFCS and Glider Platform....22
Figure 3.1: Collaborative Protocol Packet Structure....25
Figure 3.2: API Packet Structure....26
Figure 3.3: Drill Down of Communication Layers....29
Figure 3.4: Packet Sorting and Plane Instantiation....31
Figure 3.5: Received Packet Assembly....32

Figure 3.6: Construction of Transmit Packets 33
Figure 4.1: GCS Controller Settings Window 38
Figure 4.2: GCS Control Window Highlighting Collaborative Features 41
Figure 5.1: Console-VX and Netpro-VX Expansion Boards 44
Figure 5.2: Dimensions of Gumstix and Wing 45
Figure 5.3: System Hardware Layout 46
Figure 5.4: Physical Placement of MCS 48
Figure 6.1: MCS Splash Screen 49
Figure 6.2: Main Loop of MCS code 50
Figure 6.3: State Diagram for Collaborative Modes 52
Figure 6.4: Dimensions and Bearings for Path Planning 55
Figure 6.5: Sweep Width Calculation 58
Figure 6.6: Turn Style 59
Figure 6.7: Enter Distance Layout 60
Figure 6.8: Sweep Layout 61
Figure 6.9: Turn Construction 62
Figure 6.10: Simple Lawnmower 63
Figure 6.11: Interlaced Lawnmower 64
Figure 6.12: Navigation Modes in Turns 66
Figure 7.1: Initial HILS Setup 69
Figure 7.2: Rack Style HILS Setup 70
Figure 7.3: Initial Loiter Location 72
Figure 7.4: Display of Voting Output 72

Figure 7.5: T1 Base Speed Test Simulation.....73

Figure 7.6: Four Vehicle Speed Test Simulation.....74

Figure 7.7: 5 Knot Wind Test 76

Figure 7.8: 10 Knot Wind Test 77

Figure 7.9: 15 Knot Wind Test 77

Figure 7.10: Initial Positions and Designated Area 79

Figure 7.11: First Flight 80

Figure 7.12: Second Flight.....81

List of Abbreviations

API	Application Programming Interface
BC	Best Cost
BTUART	Bluetooth UART
BYU	Brigham Young University
COM	Computer On Module
CSMA	Carrier Sense Multiple Access
CTS	Clear To Send
EPP	Expanded Polypropylene
FCS	Flight Control System
FFUART	Full Function UART
FOV	Field of View
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GCS	Ground Control Station
GPS	Global Positioning System
HILS	Hardware In the Loop Simulator
IMU	Inertial Measurement Unit
IO	Input/Output
IP	Internet Protocol
IR	Infrared
MAC	Media Access Control

MC	Map Cost
MCS	Mission Control System
MILP	Mixed Integer Linear Programming
MIT	Massachusetts Institute of Technology
PID	Proportional-Integral-Derivative
PWM	Pulse Width Modulation
RAM	Random Access Memory
RC	Radio Control
RF	Radio Frequency
RX	Receive
STUART	Standard UART
SWaP	System Weight and Power
TX	Transmit
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Aerial Vehicle
UI	User Interface
VCU	Virginia Commonwealth University
VACS	VCU Aerial Communications Standard
XML	Extensible Markup Language

Abstract

The purpose of this research was to design a multiple UAV system with collaborative operation. This project is built on work that has been done in the field of Unmanned Systems at VCU and is aimed at providing a starting point for research into collaborative control of multiple UAVs. The current GCS software was extended to include multiple vehicles per single controller via a new communication protocol. Many changes were made to the user interface to facilitate controlling multiple vehicles with a single operator. A second processor, called an MCS, was added to each vehicle to allow for greater flexibility and processing power, while maintaining backwards-compatibility and limiting infringement on the real-time processing of the FCS itself. The system was fully simulated via both hardware and software simulators, and ultimately the system was field tested using multiple vehicles collaboratively searching a defined area.

Chapter 1: Introduction

1.1 Overview

Unmanned systems are very versatile tools that are becoming more and more prevalent in everyday life. The obvious advantage is that a computer-controlled system is able to go places that would be dangerous or fatal for a human operator to go. Initially, unmanned vehicles were used greatly in military applications but now have extended into many civil applications. These include applications such as agricultural monitoring, fire management, weather monitoring, and telecommunications [1,2,3]. The greater acceptance of unmanned systems has spurred more development so naturally the progression is towards more efficient or rapid operation [4,5,6]. Most configurations utilize a single UAV and operator, and while this is an effective approach there are definite benefits to using multiple UAVs to collaborate on a single task.

1.2 Motivation and Goals

The progression of research in unmanned systems is moving toward collaborative operation among vehicles. This opens up a new area where vehicles work together to provide a better solution to a problem, whether it be area search, constant surveillance, or formation flight. The completion of a new inexpensive FCS design [7], by a fellow VCU graduate student, paved the way for the movement of VCU UAV research into the area of collaborative operation. The overall goal of this work was to provide an initial platform for multiple, collaborative UAV research at VCU.

The GCS version that has been used through development usually controls a single vehicle per modem. Some testing had been done where the GCS would utilize multiple modems on the ground to control multiple vehicles but this is a less than ideal scenario. This led to a need for a system that could utilize a single ground modem and control multiple aircraft. As well, the vehicles needed to have the ability to communicate amongst themselves to support collaborative operation. This would mean a system that could utilize an addressing scheme to send data to specific nodes on a network. Thus, one of the tasks to be completed was the modification of the GCS and communications mechanisms to support multiple vehicle operations and communications

In addition to the communications requirements, each vehicle needs the appropriate processing power to execute collaborative operations. It was decided that the processing ability should be separated from the FCS to make the system distinct from the hard, real-time FCS control. Separation allows full use of each of the processors resources and the potential to run more complex algorithms in the future. This system also needed to be coded to allow additional routines to be added easily.

As well, the replication of the test system and simulation setup was needed to allow future students to quickly verify and simulate code. The final goal of this work is to actually flight test the design. Flight testing is the ultimate trial for any unmanned system, and it is this distinction from simulation that provides the most effective data.

1.3 Research Scope

This thesis will describe the design and implementation of a collaborative system for multiple UAVs based on the previous work done at VCU. It will cover modifications made to the existing VCU GCS and the inclusion of a new MCS co-processor added to the UAV. The implementation used on the MCS will be discussed in detail, including a collaborative search algorithm that defines an area to be searched by multiple vehicles. Finally, the results of simulation and flight testing of the search area operation will be presented.

1.4 Thesis Layout

The remainder of this thesis is laid out as follows. Chapter 2 discusses background research into collaborative systems and collaborative algorithms. Chapter 3 discusses the API mode utilized for communication, the networking scheme and issues involved, and a description of the modifications made to transmission and reception of data on the FCS and GCS. Chapter 4 focuses on the user interface updates, specifically those made to improve control over multiple aircraft. Next, chapter 5 discusses the MCS hardware, including its layout within the current system and physical location on the test vehicles. It will also describe the factors that were used to determine the actual MCS hardware. Chapter 6 describes the MCS software which performs the collaborative operation onboard the aircraft. This includes an in-depth discussion of the search area algorithm used to test collaborative operation. Chapter 7 discusses simulation results from both hardware and software simulations. Examples of simulations using the HILS will be shown and discussed under various conditions and test parameters. In addition, flight testing results will be discussed here. Graphs and information from the vehicles will be shown and

analyzed. Finally, chapter 8 provides conclusions from this research as well as suggestions for future work and improvements.

Chapter 2: Background

This section will provide a discussion of different collaborative research projects to give a broad view of the work in this area of study. Section 2.1 discusses different cooperative operations and control provided by a particular design. If available, a discussion of the hardware capabilities of the test system is presented. Section 2.2 discusses some of the path planning techniques researched in the field of unmanned systems that can be applied toward collaborative flight. Finally, an overview of the MiniFCS, the FCS used in this work, is presented as well as the VACS protocol that is used on all VCU UAVs prior to the development of the protocol used in this collaborative system.

2.1 Collaborative System Review

2.1.1 University of California, Berkeley

The Berkeley Center for Collaborative Control of Unmanned Vehicles (C3UV) has an impressive collaborative software system. The basis of all decisions in the collaborative planning process is decentralized, and planes update their decisions dynamically as other vehicles publish their state and plans. If there is no communication possible between vehicles, each plane will initially attempt to complete the mission alone, but as vehicles come into range, they begin to participate in performing the overall task and performance is increased.

There are multiple possible tasks that can be implemented using this technique, including: visit point, patrol segment, patrol area, and guaranteed search. The first three types of actions are variations of a single task, to visit a point, set of points, or defined region. These can be continuous or single events and are not subject to strict periodic time constraints, but are able

to be reactivated at specified times. This is due to the dynamic nature of the environment the vehicles operate in. Guaranteed search is a search task in which the vehicle is given an uncertain initial position of a target and a known upper bound of target velocity. Given the velocity of target *V*, a vehicle velocity of *U*, an initial circle area of radius $R_l(0)$, radial coordinates of the vehicle *r*, and radial coordinates of the target R_t, the optimal path of evasion by the target (γ) can be determined using [8]:

$$\gamma = \frac{Vr}{UR_t}$$

The vehicle then attempts to locate the target by determining the size of the expanding search area based on the possible escape of the target. This is done using a combination of straight search lines and spirals moving out toward the edge of the search area. Figure 2.1 illustrates the use of this equation in a pictographic representation of the guaranteed search scenario. Also included is a system to ensure avoidance of no-fly zones by any task [8].

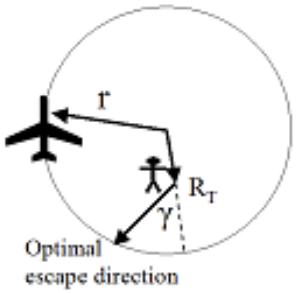

Figure 2.1: Guaranteed Search [8]

The collaborative system utilizes a Mission State Estimate to attempt to assign tasks to UAVs with low cost while fulfilling all constraints. A task is feasible if a vehicle has the lowest cost published for that task. Tasks are prioritized and added to a plane's plan if the task is

feasible and has the lowest cost. If another vehicle publishes a lower cost later, that task is de-allocated from the vehicle's plan and the update published. Based on the task and know vehicle parameters the system then defines a projected time for task completion. When a task is finally added, the Mission State Estimate for that vehicle is broadcasted and the next iteration begins. Therefore the system is constantly updating and de-conflicting if necessary [8].

The test platform used is a Sig Rascal 110, shown in Figure 2.2 with the payload inset. This vehicle uses a 32-cc two-stroke for endurance of over an hour. For collaborative operation and higher-level control there is a PC104 computer system. The autopilot that controls the actual aircraft is a Piccolo by Cloud Cap Technology [9].

Figure 2.2: Berkeley Sig Rascal [8]

A serial interface connects the PC104 to the Piccolo and payload for communication for higher-level functions [9]. The system utilizes a 900MHz radio link for control of the vehicles as well as a 2.4GHz, 802.11b wireless connection for used for inter-vehicle communication. The PC104 is a stacked design utilizing a 700MHz Pentium III processor. There are additional sensors and

equipment onboard including video capture equipment and a 1-Watt amplifier for the 2.4GHz radio link [8,10].

2.1.2 Massachusetts Institute of Technology

MIT's Aerospace Controls Laboratory utilizes a unique UAV planner system to generate solutions to cooperative tasks. The coordination algorithm combines MILP and decomposition to assign costs based on the environment and possible obstructions. The costs are then forwarded to the task assignment system to determine distribution of work. To overcome the high complexity of exact solution determination, the system uses a petal algorithm to estimate if solutions will ultimately be part of the final solution. This reduces overall complexity and thus computation intensity. Larger sets of resources require an extension in the form of a receding horizon style task assignment. This iterates over the tasks and reducing the complexity. Finally, the flight path of the UAV is constructed, avoiding obstacles, in a bounded time [11].

The autopilot used in their test vehicle is a Cloud Cap Technology Piccolo [9]. Using the Piccolo also enables use of the included HILS for real-time ground testing. Figure 2.3 shows the HILS setup used by MIT. An interesting addition to their simulation structure is that of a central data management hub. This is capable of simulating communication and networking issues. FlightGear is used as the simulation software in the test bed setup [11,12].

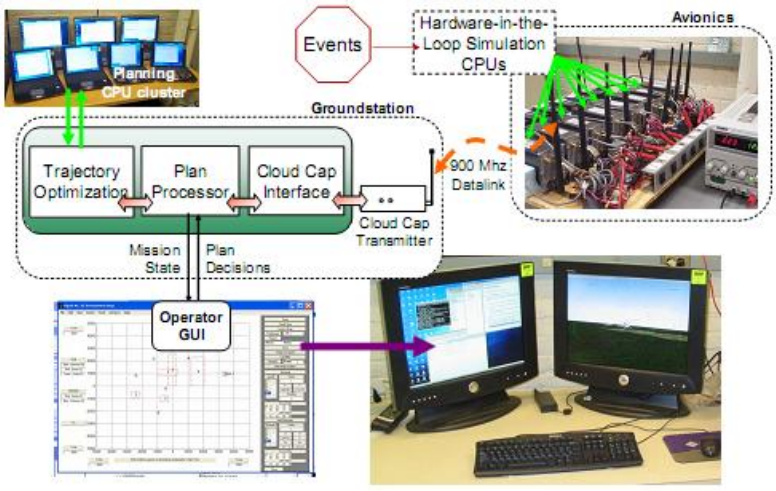

Figure 2.3: HILS setup for MIT [11]

State estimation of the vehicle and control of flight surfaces is done onboard, while planning is done off-board. The off-board control outputs the waypoints and paths to the vehicles. The actual test platform is a Tower Trainer 60, shown below in Figure 2.4. There are actually eight of these vehicles for use in MIT's collaborative research.

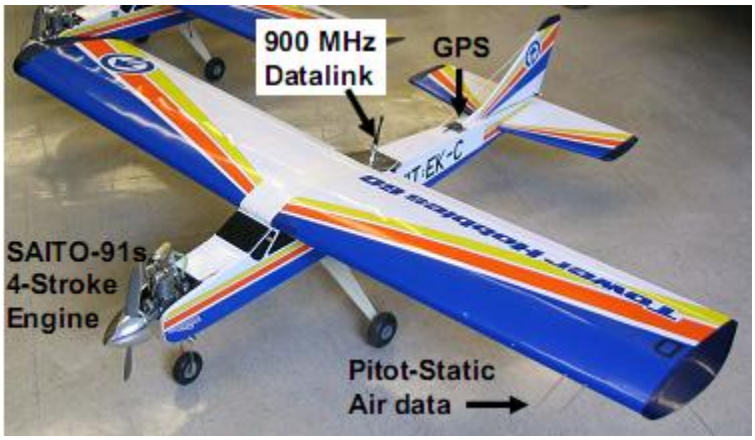

Figure 2.4: Tower Trainer [11]

The vehicle has an endurance of greater than 40 minutes and a payload capacity of 3 lbs. The command link is via a 900MHz with an approximate range of three miles. This is used by the

planner to update flight paths and commands. Payloads include video systems with control via a 2.4GHz radio link as well as magnetometers for estimates of wind [11].

2.1.3 Brigham Young University

Brigham Young University has another example of a decentralized collaborative system. Their platform is described in four basic steps: definition of objective, identification of coordination variables and function, derivation of a centralized solution, and finally construction of a decentralized solution by building a consensus among vehicles. Ultimately the system is based upon building a centralized solution and then decentralizing it throughout the aircraft.

As well, they define the tasks in different level of coupling. An objective coupling is defined as one in which the decisions of a vehicle only affect itself. Local coupling is defined when a vehicle's decisions affect the local vehicles cost analysis for a task. Essentially, the main distinction is that the local coupling affects the cost analysis of other vehicles; whereas, objective coupling is unaffected by the decisions of individual vehicles. Full coupling is defined when all vehicles must know of the decisions of other vehicles. Each member must know the decisions and trajectory of other vehicles in the team. Dynamic coupling occurs when the vehicles have physical interactions, such as formation flight. The flight path of one vehicle will affect the physical flight of another, while some planes in the formation will not be affected by others [13].

Much of the system relies on the definition of cooperative constraints, variables, and objectives. The cooperation constraint is the definition of the goal of the UAV team. A cooperative objective is a function that describes a secondary achievement such as limiting overall fuel consumption. It is not necessary for all cooperative operations to include a

cooperative objective. Coordination variables represent the data that must be shared to facilitate collaboration. The coordination function describes the effects of changes in the coordination variables. Given these variables and functions the problem is defined as a centralized strategy. If the algorithm is deterministic and the local inputs the same then the local instantiation of the problem on each aircraft will also result in the same solution on each aircraft. Otherwise, the planes must maintain sufficient range to come to a consensus and come to a final solution [13].

Applications of their methods include: cooperative timing, cooperative search, and cooperative fire surveillance. Cooperative timing is defined as the arrival of multiple vehicles to the same location at the same time. The effectiveness is measured by the difference in their ability to avoid threats and still arrive simultaneously. Cooperative search involves the flight of multiple vehicles while maintaining communication and still avoiding collision. Ideally the vehicles avoid threats but maximize the number of targets observed. Fire surveillance is a particularly interesting operation in that the vehicles monitor the perimeter of an expanding fire. Vehicles move along the perimeter until they encounter another vehicle, exchange information, and reverse direction [13].

The test platform used is a Zagi XS EPP foam flying wing vehicle that uses BYU's Kestrel autopilot [14]. This utilizes a Rabbit 3000 28MHz processor for control of the aircraft. Figure 2.5 shows the test system in its entirety and highlights the lightweight nature of this setup.

Figure 2.5: Kestrel Autopilot, Test Vehicles, and Ground Station [13]

A 900MHz radio is used for GCS command and control as well as collaborative communication. This is a low-power, 9600 baud link that uses a turn-taking scheme to avoid collisions and issues. Actual flight tests involved persistent imaging and simultaneous arrival missions with teams of three vehicles [13].

2.2 Path Planning Review

2.2.1 Raster Search Pattern

In [15], a raster path planning technique is used to search an area of arbitrary shape and consists of long straight sweeps connected by 180° turns. There are also no constraints on completion time or optimization. As well, it is stated that this algorithm is aimed at use with convex polygons. Each arbitrary area is defined by the corners.

An approach direction defines the direction of the sweep pattern. It is important that the sweep direction of the pattern generate long sweeps mostly because the longer the sweep the more effective the search pattern. The 180° turns are performed outside of the region to be

searched as the image sensor is considered to be ineffective while turning. The origin of raster is defined as the point where the raster pattern begins and the progression direction is the movement of the pattern perpendicular to the sweeps. The raster pattern shown below, in Figure 2.6, is an example of the output of this algorithm, boundaries define a region of arbitrary shape and the progression of the pattern is shown with and arrow in the south-eastern direction.

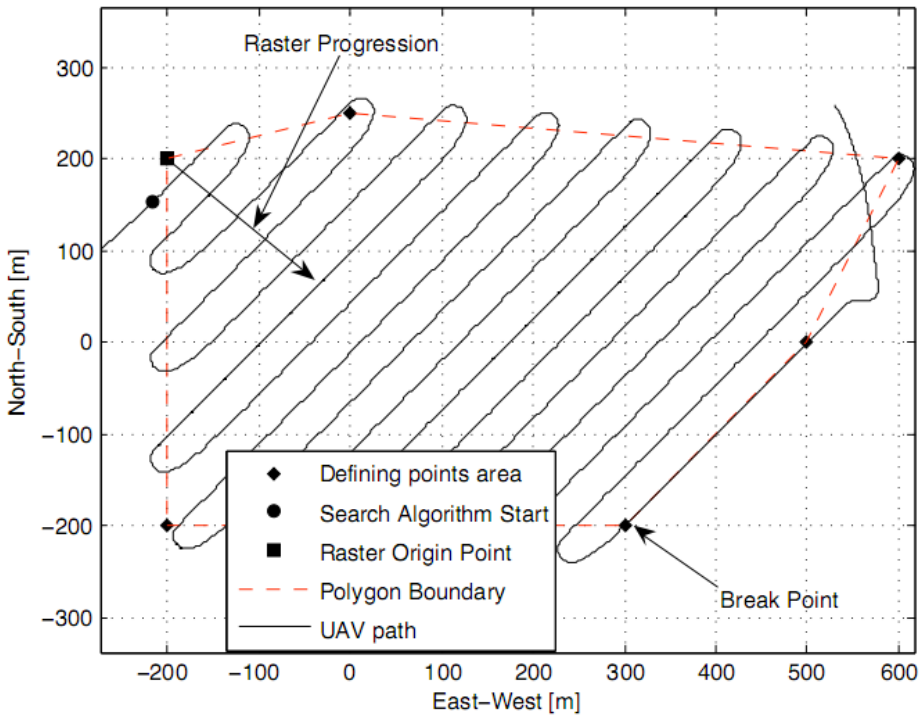

Figure 2.6: Raster Pattern [15]

The algorithm to plan the path for a raster pattern has a number of parameters that determine the actions of the code. First, the location of the vehicle must be determined to be within the area or outside. A value that defines the amount the vehicle has turned is calculated and is used to determine when to apply a smoothing parameter that commands the vehicle to exit turn mode. This is done to make the transition from the turns at the end of the sweeps into the next sweep as gentle as possible. Shadow points are used to generate the turns at the end of each

sweep line. As noted, the operator needs to utilize the operation to generate the longest sweeps possible even when this may result in searching a larger area, as it can potentially maximize search time compared to flight time.

2.2.2 Entropy Based Search Pattern

The author in [16] uses entropy to define priorities to different areas in a search region. This representation is assigned to a search area to be evaluated using a lawnmower pattern, thus a column in the defined area must have the same entropy throughout. The entropy graph shown in Figure 2.7 shows the uniform column entropy. This is a constraint because of the desired sweeping pattern used to search an area.

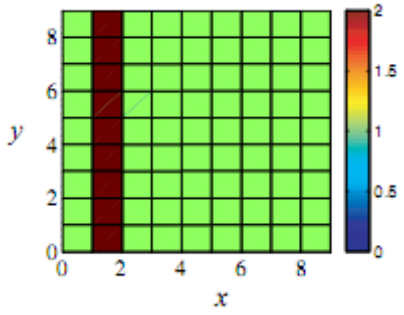

Figure 2.7: Entropy Graph [16]

A representation of the camera footprint of the vehicle is utilized as a 2x1 rectangle. When the vehicle passes over a cell the entropy is reduce by one.

Figure 2.8 shows the example grid size of 10x9. Given a typical lawnmower pattern on the grid the entropy of the area after completion can be determined. The path is shown in the left portion of Figure 2.8 and shows that the vehicle typically navigates between cells to utilize the 2x1 view of the camera. Higher levels of entropy are due to ineffective imaging when the vehicle is turning.

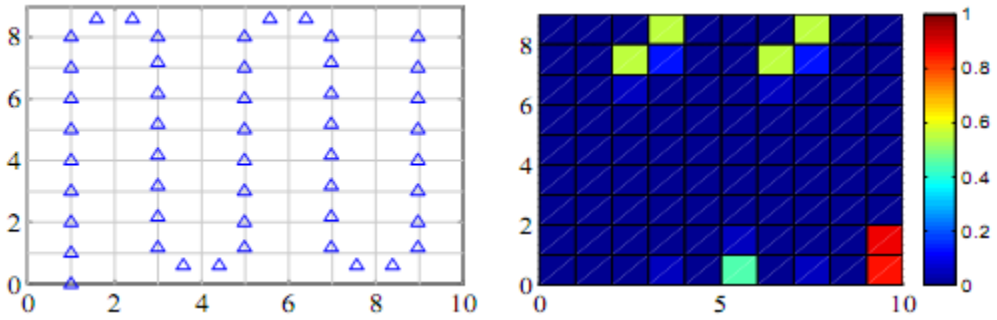

Figure 2.8: Path and Resulting Entropy Reduction [16]

The author states that a simple solution would be to move the turning regions beyond the area to be searched. However this does increase fuel consumption and total mission time.

Given the entropy of lanes alongside a particular path a number of revisits to each path can be determined. This is based on determining the entropy of the lanes that run alongside a defined path. Once the number of revisits has been determined the path must be generated. This is based on an assumed turning radius of two cells, the same as the width of defined camera view. As well, the other constraint is to try to find a minimal travel time. This is represented by a vector of paths with defined turning difficulties, C_{turn}. The vector shown is a representation of the difficulties from moving from the initial column to other columns [16].

$$C_{turn} = \{\mathrm{H\ H\ 1\ 2\ 3\ 4\ 5\ 6}\}$$

Here *H* is a variable that represents a large number greater than one, or an impossible turning scenario. So it is not possible to turn onto the same path or the path immediately adjacent. As well, it shows that the difficulty increases as the paths become farther apart. The order of paths is iterated over until an optimal path is found. Given the example shown before with a 10x9 region with entropy of one, the final path permutation result is shown.

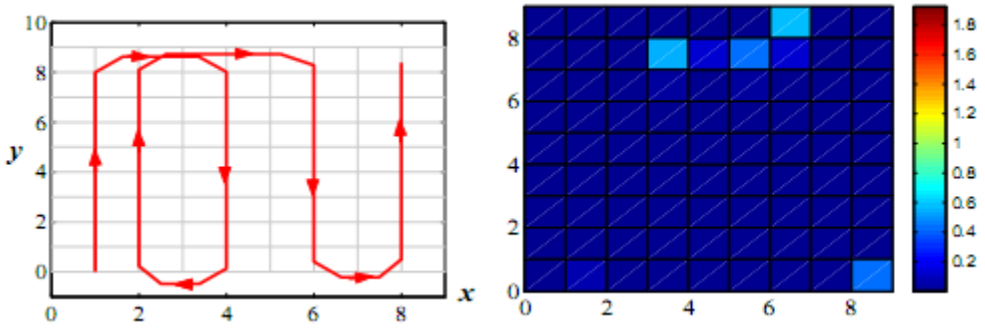

Figure 2.9: Final Path and Resulting Entropy Graph [16]

The final plan, shown in Figure 2.9, moves from paths in the following order: 1, 4, 2, 6, and 8. Simulations are said to provide more than 97% coverage of a region using this method [16].

2.2.3 Voronoi Graph

A graph is defined in [17] as a series of vertices and edges. These can be used to solve path planning problems by assigning weights and costs to edges connected by vertices with 3-D coordinates. This can be used to generate paths, in this example, to avoid radar sites and provide a stealthy corridor of travel for a UAV.

Given a set of know radar sites a series of points can be generated between each set of three radar sites. These points are at the centers of circles that are drawn connecting three points (radar sites) but do not intersect other radar sites. The points at the center of each circle are the Voronoi points. The set of all triplets of radar sites are called the Delaunay triangulation [18]. Edges from the Voronoi points are only drawn if the edges are shared with a Delaunay triangle. Weights can be applied to the edges that define the length of the edge and the detection by radar sites. This presents a simple path for movement through a series of radar sites or any arbitrary obstacles [17].

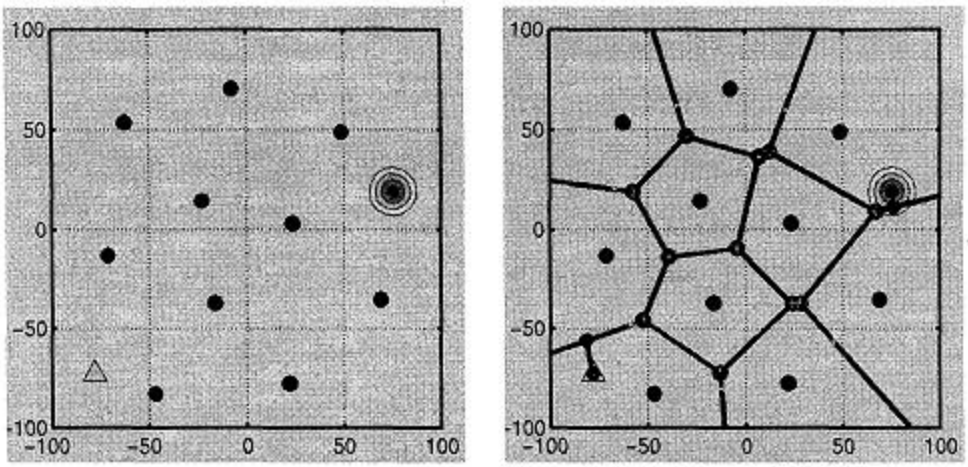

Figure 2.10: Voronoi Graph of Radar Sites [17]

Figure 2.10 shows an example graph of radar positions in a defined region. Radar sites are the black dots, the vehicle is the triangle, and the destination is the radiating dot. Using the technique described a series of paths are constructed flying between radar sites. These paths can be used to find an optimum path from the vehicle to its desired destination.

2.2.4 Robust Real-Time Route Planning

In [19], the authors discuss a method for real-time route planning based on a set of defined criteria. These criteria include: the route does not exceed the physical limitations of the vehicle, exceed the workload for a pilot, and violate any mission parameters. The restriction based on the pilot would of course not apply to unmanned systems, but the planning discussed here is designed to be general to route planning for many different applications.

Many constraints may exist in a particular path such as route distance, route length, or turning angle. As well, the authors mention that these parameters may be considered to be adaptable or changing during the course of the mission. In this case, the optimal solution is NP-complete and therefore restrictive for real-time scenarios. The algorithm utilizes a grid-based

system where a “start” and “goal” cell is defined for the map. A cost for each cell is calculated and can be based on any number of parameters. The set of costs for each cell results in a set of costs the same size as the grid called the “Map Cost”. Next, a “Best Cost” path is generated by moving from the cell to cell via the minimum cost in the eight surrounding cells of any cell. A “walking path” can be determined by finding the lowest BC costs from the start cell to the goal cell. Figure 2.11 illustrates the grid created with a walking path shown in the circles. Each circle contains a “Map Cost” and a “Best Cost” value for path determination.

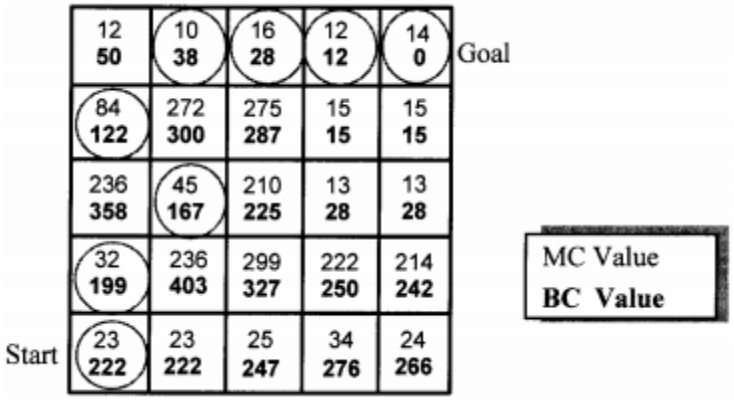

Figure 2.11: Cost Map [19]

Using the BC map generated initially, additional constraints such as minimum leg length and route length can be applied. The authors utilize their new approach called Sparse A* Search (SAS). This is based on the A* heuristic searching algorithm, which generates cost functions for various paths in a space by adding points to the path based on the lowest cost. This is used to generate even more possibilities for the path. Minimization is evaluated using:

$$f(x) = ag(x) + bh(x)$$

Parameters for *a* and *b* are weights associated with the actual cost from the start position (*g(x)*) and the estimated cost from the intermediate position to the end of the path (*h(x)*). However, the complexity of this method is greatly dependent on values of *a* and *b*. This is the reason for the use of SAS to remove areas of the search space that would make the convergence time prohibitively long. This data is used to generate a tree of paths that utilize the minimum cost from the start to goal, shown in Figure 2.12. As well, the fan out shapes can be of different sizes to facilitate changing minimum turn angles and leg lengths [19].

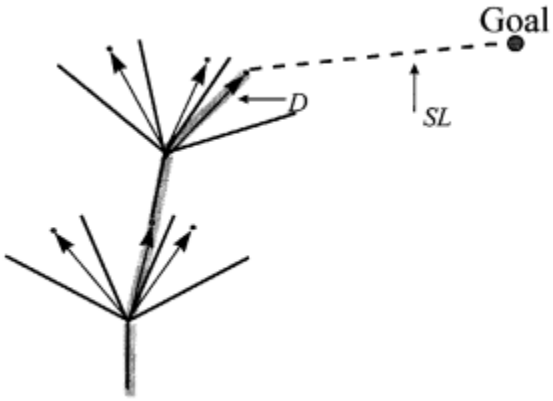

Figure 2.12: Fan out of Optimum Path [19]

Total route distance can be constrained by calculating the length of the computed path up the current point. Then the new point is only added if the addition of the point and the straight-line distance to the goal is less than the desired total route distance. There are also additional portions of the algorithm that allow a final vector of approach to the goal to be defined [19].

2.3 VCU System Overview

The VCU autopilot system has evolved greatly over the course of its existence. It began as a senior design project within the engineering department built on an Atmel FPSLIC platform

[20]. This design progressed into the next platform, which was an Atmark-Techno Suzaku FPGA [21,22]. This platform also included additional sensors for more precise attitude evaluation. This system was later improved upon by expanding the flight control software and adding support for different platforms such as jets and rotorcraft [23]. Within the past year, two graduate students have completed new FCSs deemed the MiniFCS [7] and Next-Gen FCS [24]. The Next-Gen is designed to be a high performance system and the MiniFCS is aimed at low-cost, specifically to support this work and continued collaborative research.

2.3.1 MiniFCS

The MiniFCS was designed and built by a fellow graduate student at VCU. The MiniFCS is an Atmel 32-bit microcontroller based solution constructed around used in a small glider platform. This system was also designed to be low cost to allow for multiple vehicles to be built without significant financial risk. Figure 2.13 displays the PCB layout of the MiniFCS; its dimensions are only 1.8x3.2 inches.

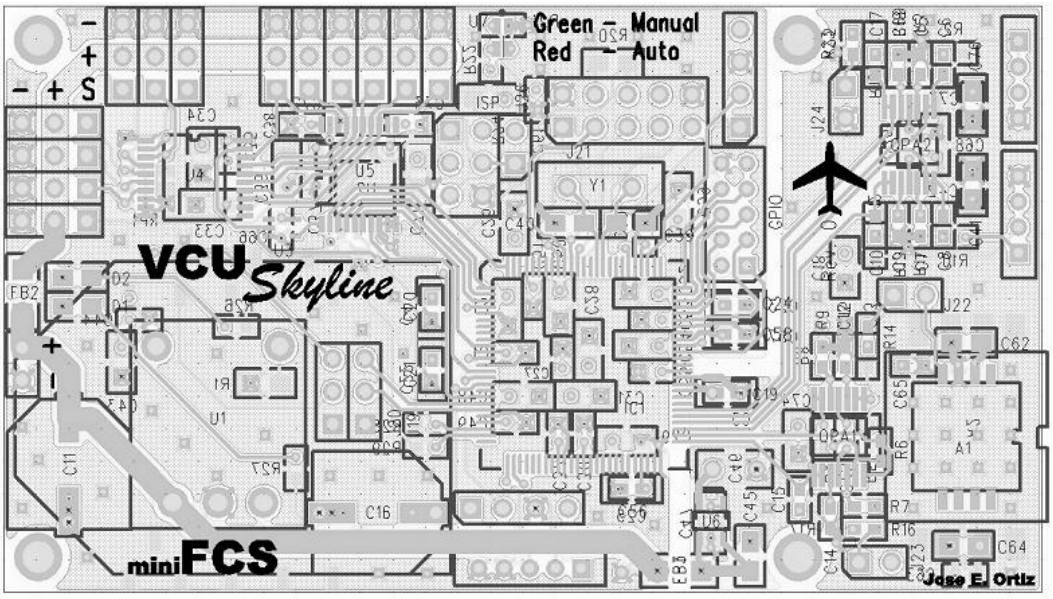

Figure 2.13: MiniFCS PCB layout [7]

The MiniFCS uses a myriad of sensors to gather data about the state of the aircraft. The system also included a new field-waypoint navigation mode.

Sensors onboard include barometric pressure for measuring airspeed and altitude. As well, GPS telemetry data can be extracted via a GPS module connected via a serial port to give accurate position data at up to 10Hz. The system utilizes IR-thermopiles to measure a relationship between the differing temperatures of the earth and sky to extrapolate pitch and roll angles. The main control link is via 900MHz radio with a 2.4GHz RC control link. The control via the RC link can be activated at any time by the safety pilot via a built in safety switch [7].

Field waypoint mode is based on adjusting course based on directional field lines. This method is designed to reduce cross-track error when traversing between waypoints. A heading vector is generated based on the position of the aircraft, destination waypoint, source waypoint, and a designate cross-track lead parameter.

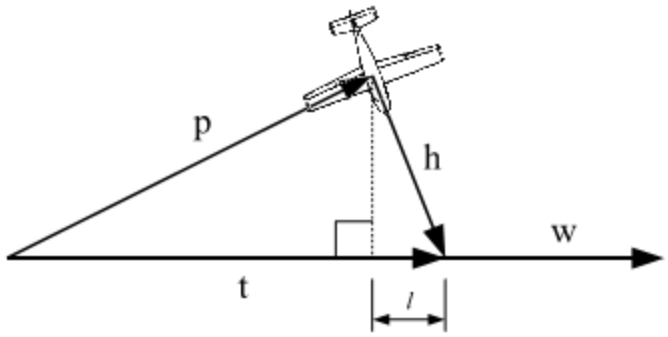

Figure 2.14: Cross-track Error Vector Diagram [7]

The vector *h* in the figure above is the final heading vector that is determined by the system and used to generate a target heading used by the control loops. Vector *l* is the lead parameter and is controllable by the operator. Cross-track lead is essentially a value that controls the steepness of the vehicles entry to the rhumb line. Control is achieved via cascaded, PID control loops. These

control latitudinal and longitudinal flight by calculating errors generated from target values and actual measured values collected from sensor data [7].

The MiniFCS also was designed in tandem with a HILS to be used for higher-fidelity testing of the system. This device allows for quicker development and testing of the system as well as faster validation of newly constructed systems. This platform was built around a Suzaku SZ130 FPGA which works with FlightGear software to generate sensor values for the FCS as well as read in PWM values for feedback [7].

The platform used is a Multiplex Easy Glider Pro, shown below. The image in the right of Figure 2.15 shows the tight quarters for the MiniFCS and necessary electronics. This vehicle is the platform utilized in the collaborative flight testing done in this work.

Figure 2.15: MiniFCS and Glider Platform [7]

2.3.2 VCU Aerial Communications Standard

The MiniFCS, like all other VCU FCSs, utilizes the VACS protocol for communication between the GCS and UAV. This protocol is used to define parameters within each packet and give the communication an organizational scheme. As shown in Table 2.1, parameters included in each packet are: two synchronization bytes, destination ID, source ID, message ID bytes, length bytes, payload, and a 16-bit checksum.

Table 2.1: Layout of VACS Packet

Sync 1 (0x76)	Sync 2 (0x63)	Dest.	Source	Mess. High	Mess. Low	Length High	Length Low	Data	Check High	Check Low

The synchronization bytes serve as packet delimiters and help the parser determine the start of incoming packet data. Source and destination bytes define the endpoints of packet traffic. Typically the GCS uses the reserved value of zero and because the system is usually only using one vehicle many FCS have a hard-coded ID of one. Message ID is a two byte field that specifies the type of data that is encapsulated within the packet. This is used to reference the plane definition file or definitions held within the FCS to determine the format of the data within the packet's payload. The length byte refers to the number of bytes of payload data that exist between the low length byte and the checksum bytes. There is a maximum of 1024 bytes possible in the payload field. The VACS uses a 16-bit Fletcher checksum for data integrity and is contained in the last two bytes of the packet.

Chapter 3: Networking, API, and Fragmentation

3.1 Digi API Mode

The MiniFCS utilizes an XBee-PRO 900 RF module for communication to and from the GCS. Previous to this collaborative effort the modem operates in transparent mode or essentially a connection between the nodes just as though there was a physical connection. Because of the modems integral role in the current system's design it was decided to continue use of these devices and utilize a specific set of instructions available in the firmware known as API mode. This mode is activated by a simple firmware change. API mode is available in the XBee and XTend series of modems produced by Digi. This is important because other VCU UASs utilize Digi modems, specifically the Next-Gen FCS which uses the XTend model [24].

API mode allows for a host of networking features to be utilized, most important of which is the ability to define destination addresses for specific data sets. Addressing schemes exist in 16-bit and 64-bit variants but are only available in 64-bit addressing schemes for the XBee series. API operation also defines all packets with the source address included and allows for remote configuration of modems using specific packet types. The API mode also allows for point-to-multipoint and mesh networking topologies.

There are a few caveats to utilizing this mode. For example, the API format of packets is not the same across all models. While the changes are minor, it still requires modification of the routines used to define and extract data from the API structure. As well, only the 64-bit variant of addressing is available for the XBee-PRO 900, and therefore requires eight bytes of data in each packet simply for the address. A 16-bit version would still vastly exceed our addressing requirements for the planes, and reduce the overhead of packet data. In addition, the API mode

has limitations on the length of packet payload. The XBee-PRO 900 used on the planes can only contain 72 bytes of data per packet; however, the API mode for the XTend modems can utilize 16-bit addressing and contain up to 2048 bytes of data per packet [25]. The payload limitation requires modification of the packet structure and is discussed in a later section.

3.2 Packet Structure

It is simplest to think of the API as an abstract layer outside of direct communication with the UART on the RF module. For example, to send data a transmit request packet would be sent to the module using the specified format. Figure 3.1 below shows a summary of the layout of the packet structure used to transfer data between planes and the GCS. Essentially, there is API structure data (blue) that every packet must conform to as part of the existing API protocol. As well, the ID specific data (red), as defined by the API protocol, must be included. Then the frame specific data (green) exists within the API structure to support fragmentation of data. Finally, the actual payload exists within the frame (purple).

Figure 3.1: Collaborative Protocol Packet Structure

The API protocol follows a standard structure with a delimiter that starts each packet, followed by two length bytes, a payload field, and a single checksum byte. This data exists in every packet as defined by the protocol. The checksum is a summation of all the data bytes in the packet, excluding the delimiter and length bytes, subtracted from 0xFF. This means that the receiving node need only add the packet bytes, again excluding the delimiter and length, to the checksum and compare the sum to 0xFF.

Figure 3.2: API Packet Structure [26]

Each API packet includes an API-specific structure, as shown in Figure 3.2, which is defined by the protocol and is denoted by a specific identifier. These identifiers are displayed in Table 3.1 and show the possible types of packets that can be sent using the API mode. In this design, only the Receive Packet and Transmit Request are utilized by the planes and GCS for communication.

Table 3.1: API Frame Types and Byte-Value

API Frame Type	API ID
AT Command	0x08
AT Command - Queue Parameter Value	0x09
Transmit Request	0x10
Explicit Addressing Command Frame	0x11
Remote Command Request	0x17
AT Command Response	0x88
Modem Status	0x8A
Transmit Status	0x8B
Receive Packet	0x90
Explicit Rx Indicator	0x91
Node Identification Indicator	0x95
Remote Command Response	0x97

Table 3.2 displays a Transmit Request format, as it would be used in the current design, including the initial structure layout above. As shown, the format includes data such as a frame ID (table 3.1), destination address, network address, broadcast address, and other options. Data values such as the broadcast radius, network address, and transmit options remain mostly unchanged in this system due to their applications to unused portions of the firmware.

Table 3.2: Example for transmit request packet layout

Transmit Request Layout	
API Packet Field	**Packet Byte Index**
Delimiter	0
Length MSB	1
Length LSB	2
Frame Type	3
Frame ID	4
Address Byte MSB	5
Address Byte	6
Address Byte	7
Address Byte	8
Address Byte	9
Address Byte	10
Address Byte	11
Address Byte LSB	12
Network MSB	13
Network LSB	14
Broadcast Radius	15
Transmit Options	16
RF Data	17
Checksum	Length - 1

Because this system is built on the underlying VACS protocol (1024 byte payload), and that protocol exceeds the payload limitation of the API mode (72 byte payload), a new communication protocol was defined to support packet fragmentation. This system takes VACS payloads that would normally be larger than the 72 byte payload limit and splits them up into frames. This frame level messaging requires new data fields that allow for re-assembly of the frames on the receiving node. These fields are included in each frame and are shown in Table 3.3. Frame design and layering is loosely based a new VACS protocol standard [27]. These values will exist within the RF Data portion of the API packet structure.

Table 3.3: Frame packet structure

Frame Field Name	Frame Byte Index
Source Tail	0
Destination Tail	1
Data ID MSB	2
Data ID LSB	3
Last/Offset	4
Frame Size	5
RF Data	6
Checksum MSB	Frame Size - 2
Checksum LSB	Frame Size - 1

The source and destination tail values refer to the tail number associated with the plane that is in communication. The data ID is a value that is incremented with each dataset that is sent. This means that multiple packets can have the same data ID value, and that value can be used in assembly of the fragmented frames. The last/offset value represents two attributes of the frame. The "last" descriptor shows whether or not the frame is the last frame in the dataset. This is signified by a one in the eighth bit of the byte value. The remaining seven bits signify the offset of the frame, or its location in the procession of frames that make up the complete data set. The frame size signifies the number of bytes in the payload that exist between the frame size byte and the API packet checksum byte. The RF data is the final payload data that is being passed between host and destination, preceded by a 16-bit message ID value that tells the receiver the format of the payload. The 16-bit checksum shown at the end of the frame was an addition made to ensure that all actual FCS generated data was validated. Because the checksums within the API packets are validated and generated by the modem, the possibility exists that it could generate a valid checksum for data that is not correct. An example of this is when data is sent, the program will generate a Transmit Request to the modem, which includes a destination address for that data. The modem receives the transmit request and sends the data,

where the receiving modem will generate a Receive Packet for the end node which is parsed by the program. This Receive Packet will include a source address, which was not included in the original transmission generated by the program. This means that the checksum byte produced by the sending program isn't the same as the checksum received by the end node parser. Thus at 16-bit checksum is utilized by the frames to ensure that actual FCS generated data is valid, and to increase the flexibility of that code making it useful in a packet structure different from API. The message ID and 16-bit checksum may not actually be within the same frame as shown in the table. These data values are generated per dataset, not per frame. It is still possible that both the message ID and application checksum would exist within the same frame, if that particular message type has a smaller size than the frame size. So the 16-bit checksum acts as an application layer check, used to ensure that the full data is valid, while the API packet checksum is used to ensure each packet, or partial dataset, is valid.

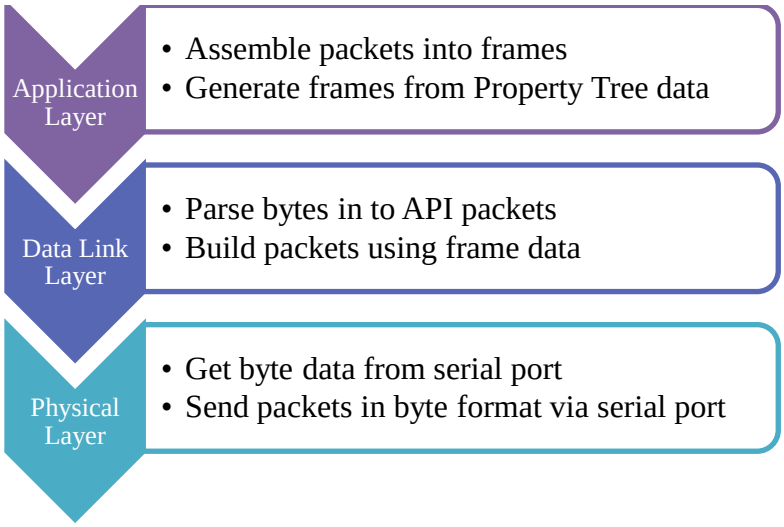

Figure 3.3: Drill Down of Communication Layers

This distinction of application layer is made throughout the program on both the GCS and plane parsing programs. Essentially code is segmented into either an application layer or data link layer function, as shown in Figure 3.3. Each XBee has a hard-coded serial number that doubles

as its address, so this is used much like a MAC layer address would be used in a typical IP network. As mentioned above, the frames also include source and destination tail numbers. Each plane is designated a unique tail number from 1-254, where zero is reserved for the GCS and 255 is reserved for broadcast messages. The tail number is used to sort packets for different planes and to ensure the desired destination is achieved.

3.3 Data Transmission and Reception

Reception of data begins with the collection of incoming data in the form of a byte stream from the serial port. This byte array is then sent to a parser that extracts valid packets from the stream. The parsing system is essentially a finite state machine that moves through the array looking for the correct API format, starting with the delimiter (0x7E). This is illustrated as the first step (green) in Figure 3.4. When the data is determined to be valid it is added to a list of completed packets. These packets are then sorted based on the tail number found in the frame. When a tail number that isn't currently known to the GCS is found, a new data class for that vehicle is instantiated and the tail number, plane type, and modem address are stored within. The packets are stored locally in the data class that belongs to the corresponding vehicle. This process is complete (red) when all the packets in the list have been sorted and necessary vehicles created.

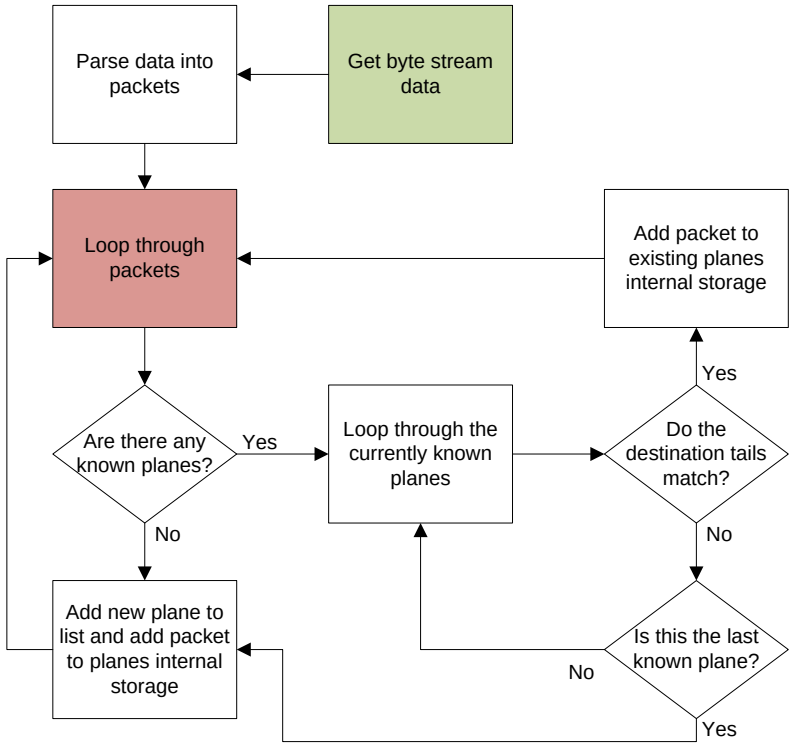

Figure 3.4: Packet Sorting and Plane Instantiation

Next the packets are assembled into valid plane data. This process is outlined in the flow chart displayed in Figure 3.5. At this point each packet contains a single data frame. The assembly works by creating temporary storage for frames as they are processed, assuming that additional frames will generate a complete data value. The temporary storage structures are designated "Fragmented" where the completed are aptly designated "Completed". If there are currently no plane data structures created, for instance at the initial creation of a plane, a temporary plane data structure is immediately generated and the frame added. As new frames are being processed they are compared against the data IDs of existing "Fragmented" structures. Each time a frame is added to the "Fragmented" structure, its offset and last frame flag are checked and a Boolean representing its addition is marked true. When the last frame flag is set then that frame's offset is used to determine the total number of frames that are needed for the structure to be complete.

If the plane data structure determines that all the needed frames have been acquired then the structure is added to the "Completed" list. This list is what is ultimately returned from the frame assembly function, as designated by the red box in Figure 3.5. The list of "Fragmented" data structures remains persistent between function calls, but any that exists beyond a certain timed threshold is removed.

Figure 3.5: Received Packet Assembly

Transmission is a very similar process, just in reverse. When a request or command is generated by the user from the control form, the data payload and message ID are assembled and

stored in the same type of structure for storing plane data in the receiving process. The initial process (green) in Figure 3.6 shows the collection of data from the property tree. The data from the property tree is gathered with the entire payload, and thus must be fragmented. The data is split up into frames based on the frame size, and attributes such as tail number, last frame flag, and offset are set accordingly. Each time a frame is generated it calls a data link routine that adds the necessary API structure and addressing for a Transmit Request message and ultimately adds the completed packet to a list that is returned upon completion of the routine. Once the data is packetized it can be sent out of the serial port to the desired vehicle as shown by the red box in Figure 3.6.

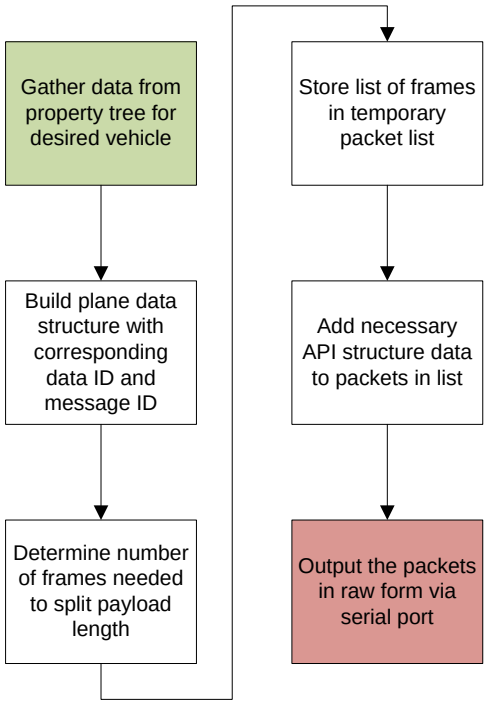

Figure 3.6: Construction of transmit packets from property tree data

The processing of API packets is essentially the same on both the vehicle and the GCS with small changes made because the GCS is written in C# and the plane in C. On the GCS the

functions for the application layer and the data link layer are written as an interface, which acts like a template. Within the GCS code there are classes for different modem types that inherit the interface and thus must define the routines for assembly and generation of frames and packets. For instance, the class that represents the modem used in this design is called "XBee64" designating it as an XBee modem using 64-bit addressing. The GCS doesn't actually have to know what functions are being called to parse and assemble data, only that there exists an application and data link layer interface to these functions. This was done to maintain flexibility in spite of the incompatible nature of the API structure among modems.

3.4 Arbitration Schemes

Given the point-to-multipoint networking scheme the communication method has some issues. It was discovered early in development that data was colliding when multiple planes were turned on. Much of this has to do with what is known as the hidden node problem [28]. Essentially, each plane has no way of knowing when another plane will be sending data and thus cannot avoid sending data simultaneously. Some schemes such as CSMA and other arbitration mechanisms are aimed at listening on the radio band and waiting until it is clear to send data [29]. There is supposedly a similar function on the Digi modems called Clear Channel Assessment, but is not something that can be observed through modem function. As well, this may prove to be a problem because it would simply cause starvation of nodes, where the first node that transmits never relinquishes access. It could also be ineffective because in this scenario the nodes would be mobile, and the planes may only find the channel to be clear because they are out of range of each other while still in range of the GCS.

There is a clear relationship between the amount of traffic and the level of collisions so the simplest solution is to simply reduce the output rate of each plane until the collision

percentage is reasonably reduced. In this design the rates of all data from the plane were reduced to 2Hz to lower the overall collision rate. There are other arbitration mechanisms such as polling and token passing that could be utilized as well, and a variation of polling was tested and compared to the reduced rate method. Essentially, the polling method tested by sends time slices, or packets with a specified time, to the planes which represent the amount of time they can communicate. The duration of the time slice was generated using:

$$sliceTime = 1/(pc * r)$$

In this equation, *pc* is the plane count and *r* is the desired update rate. Initially, *r* was typically 2Hz since each plane updated at that rate but other tests were run with higher rates. When a plane received this packet it would begin transmission for the allotted time. The GCS would also wait the allotted time plus a propagation delay estimate, and when the time elapsed the GCS would send a time slice packet to the next plane. As a safety measure, if the planes didn't receive a time slice packet within a set timeout, they would reset to reduced rate mode and send all data at 2Hz to eliminate losing link with a plane that missed a time slice packet.

Table 3.4: Polling Mechanism Results

Update Rate (Hz)	Time Slice (ms)	Sent	Received	Lost	Loss (%)
4	62.5	50235	50220	15	0.02986%
5	50	50134	50098	36	0.07181%
6	41.66666667	61674	61557	117	0.18971%
10	25	50578	50311	267	0.52790%
None	None	101675	100373	1302	1.28055%

Table 3.4 refers to a pure data test, where the planes were simply run and then stopped to measure the difference in packets sent by the MCS and packets received by the GCS. The effect of too high an update rate is seen as the higher rate of 10Hz approaches the packet loss without

any polling. This appears to be a result of the overhead generated in packet transmission to the planes, or all the time slice packets that now must be sent. Now the distinction of this test from the test results in table 3.5 is that the planes were not asked to operate collaboratively and therefore, have even less traffic. When collaborative operation occurs there is the combined traffic of having to send data to the GCS and now communicate with other vehicles in the network. Table 3.5 shows the results from polling versus reduced rate during collaborative operation.

Table 3.5: Collaborative Operation Polling Results

Update Rate (Hz)	Time Slice (ms)	Sent	Received	Lost	Loss (%)
4	62.5	53762	53207	555	1.03233%
None	None	50314	50176	138	0.27428%
4	62.5	248070	239579	8,491	3.42282%
None	None	300581	299025	1,556	0.51766%

The shaded groups represent sets of data that were flown on similar patterns. There is some packet loss because of transmission between vehicles that is not counted, but this would be insignificant (less than 100) compared to a difference of 6,935 packets found in the second test. It would seem that the additional overhead and restriction on the planes communication is somewhat counterproductive when the vehicles must engage in collaborative operation.

This presents another issue involved in wireless networking because transmission is half-duplex. Before, with the planes all running free using the reduced rate, the GCS spent majority of its time receiving data and only transmitted data when the user asynchronously made a command or request. When polling, the time slices must be sent out many times a second, and this increases with more planes to maintain an effective operator experience. This generates a lot of transmission overhead and leaves little room for error where the GCS must determine the end

of a plane's time slice. If the GCS ends too soon then the chances of collision with incoming data go up, and typically the new data the GCS is trying to send is the time slice for the next plane. Polling has some advantages in that the entire bandwidth of the link can be utilized between the GCS and each plane as opposed to the link being divided between the number of planes. This may allow for future developments to increase the number of planes while increasing the transmission rate of each plane. This method is currently not being used but may be a building block for larger groups of planes or more advanced arbitration schemes.

Chapter 4: GCS and Interface Updates

Many other attributes of the system had to be modified to allow for multiple vehicles utilized by a single modem or plane controller process. The system of handling requests and commands from the GCS operator had to be adapted to determine which plane was being designated. As well, a new class for storing individual data about vehicles helps to maintain information and organize data transmission and reception. A new control form shown in Figure 4.1, based upon the old, contains additional options for selecting frame size and modem type as well as statistical information given the myriad of new data. The figure below shows the same form but with different tabs open to show all the data available to the user. As shown, the modem and frame size can be selected by the user. The frame size is typically set to 64 but can be lowered; however, this only needlessly generates additional overhead and is not recommended.

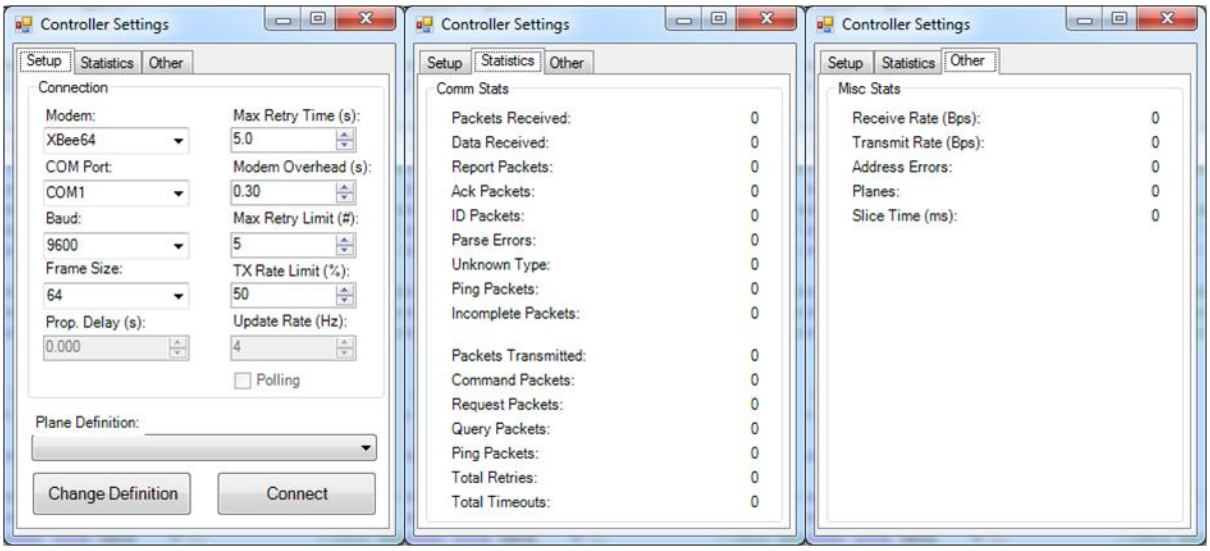

Figure 4.1: GCS Controller Settings Window

As noted before, additional classes can be added to support other modems that utilize API mode. The form will automatically search the assemblies and find classes that inherit the data link and application layer interfaces and add them to the "Modem" drop down box, making future developments more streamlined.

The property tree structure does not currently allow for a clean removal of planes when they are added, but the plane count needs to be dynamic so that collaborative operations can utilize available planes only. This is done by linking the total plane count to each plane's activity. In essence, planes are removed from the count when they are no longer active, or have passed an activity timeout value without data reception. Once they begin data transmission they are added back to the total count. Due to the nature of collaborative operation, and the high probability of re-using planes, the inability to fully remove planes from the property tree structure seems like a minor obstacle if any. Basing the plane count on activity level allows for planes to be added and removed to support future collaborative operations where cycling of available assets may be utilized.

The GCS uses XML documents to layout messaging formats, and this means that the additional collaborative operation needed a set range of messages. Thus far the range is set from 300-399, where 300-349 are used for collaborative messages that are between the plane and GCS, and 350-399 are used from plane to plane communication. The ID field is 16-bit so there is plenty of room for expandability in the future, but this range was determined to be unused by any other current plane message definition.

The collaborative operation is initiated by sending a search area command to the plane via the GCS. Changes were made to the control window to allow drawing of a rectangular area that defines the region to be searched. This is achieved by selecting the two corners of the

rectangle, first by clicking on the first corner, then moving to the opposite corner and double-clicking. The mouse wheel is used for rotation with the initial corner being the point of rotation. Once the region is applied it is sent to all vehicles currently known by the GCS. This message includes the four points of the rectangular region and the number of planes that the GCS has identified. Typical waypoint data includes arrival range, airspeed, altitude, and other parameters, but, to reduce packet size, each waypoint only consists of latitude and longitude.

Figure 4.2 shows the additional UI elements added that show the status of the vehicles and the vehicles' tail number and color. A box that displays any discontinuities in the collaborative process appears when an error has occurred and a context sensitive button gives the user options for handling the issue. In the figure, an example of a voting discontinuity is shown. The red box that displays “Vote” would normally be invisible and the context button that reads “Re-Vote” would be disabled. As well, there is always the option to return a vehicle or all vehicles to their location before collaborative operation was initiated.

Search path points are collected in a series of sets to reduce instantaneous traffic on the link. Because of the nature of the collection of waypoints from the collaborative search path there needs to be a single user retrieving the waypoint sets. There is the potential that a second operator can view the control window via a network connection. In this scenario, the control windows will both issue commands to retrieve the flight path causing needless traffic as well as potentially conflicting results.

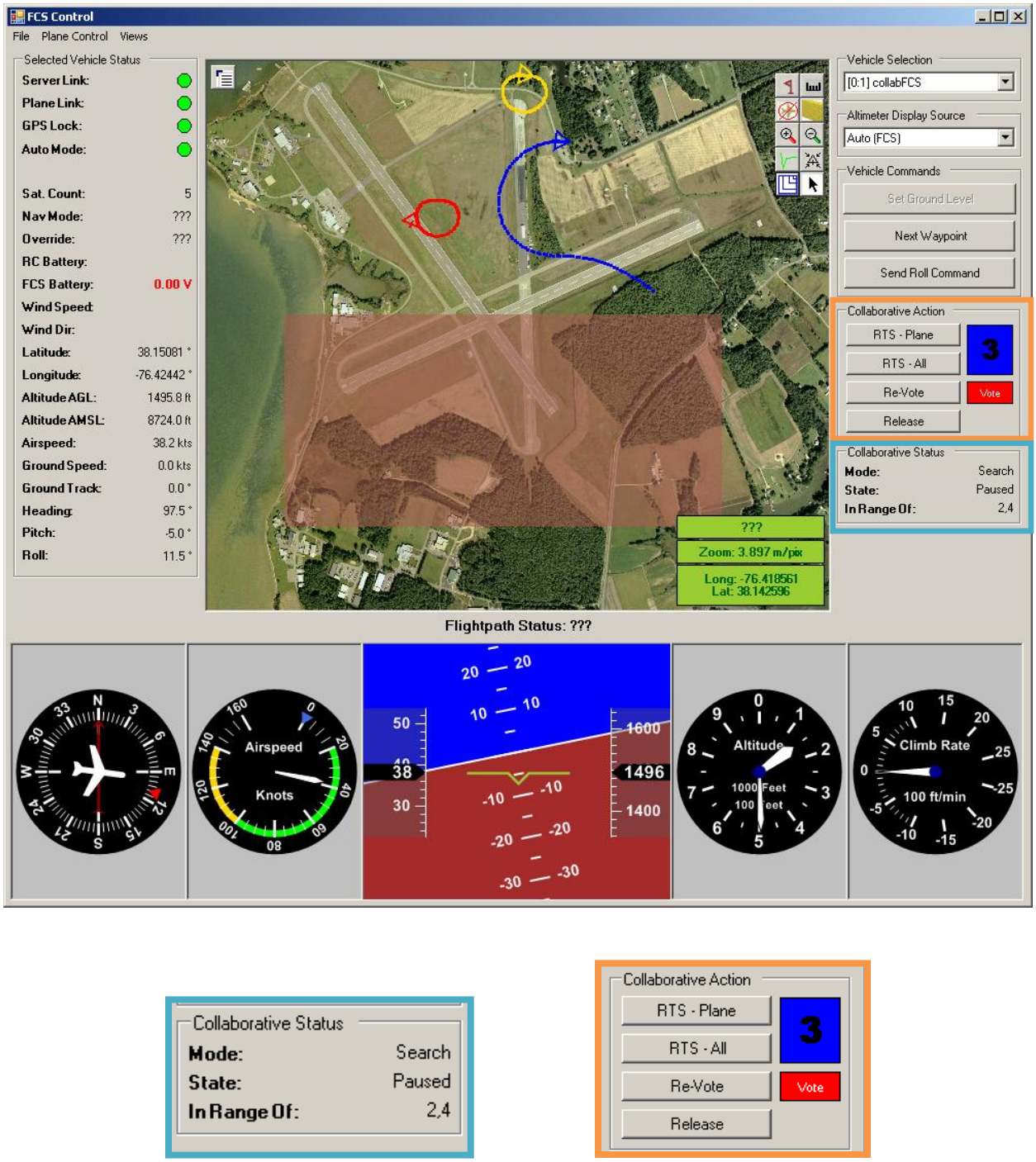

Figure 4.2: GCS Control Window Highlighting Collaborative Features

To circumvent this, the first control window to open registers its computer name and handle to the property tree. When a control registers to the property tree, the button on its UI

reads “Release”, to signify the user’s ability to relinquish control of search path collection. Other UIs will read “Locked” on a disabled button to show that another control application is responsible for data collection. The property tree is persistent across the control windows and thus allows a single control window to collect waypoints from the vehicles. Once the entire collection is gathered by the control it posts them on the property tree to allow all other control windows to view the path.

Chapter 5: Mission Control System Hardware

5.1 MCS Hardware Description

The Mission Control System, or MCS, is the portion of hardware that handles any collaborative operation. This hardware is used to allow the FCS to maintain its hard real-time constraints and still expand functionality of the current system. This design utilizes a Gumstix Verdex Pro XL6P COM [30]. This device contains a 600MHz Marvell processor with 128MB of RAM and 32MB of flash all on a compact form factor.

There are many reasons for the selection of this particular platform over other options. First, the raw performance of a 600MHz processor in this size ensures that it will fit and perform easily in the test bed utilized in this research as well as those to come. A Micro-SD slot also gives the ability to log additional data on systems, like the MiniFCS, that do not contain a significant source of storage. Currently all MCSs have an 8GB Micro-SD card for logging data and statistics. Another advantage is the logic level voltage is the same as what is used on the MiniFCS, 3.3V, making it readily compatible; as opposed to another Gumstix series, Overo, that utilize 1.8V logic level [31]. Because this system relies heavily on serial communication the logic level and the three available serial ports make this platform easier to use. The serial ports are available via a Console-VX expansion board. There is also a Netpro-VX expansion board, shown in the right of Figure 5.1, which allows connectivity via Ethernet which greatly speeds up development and updating of software.

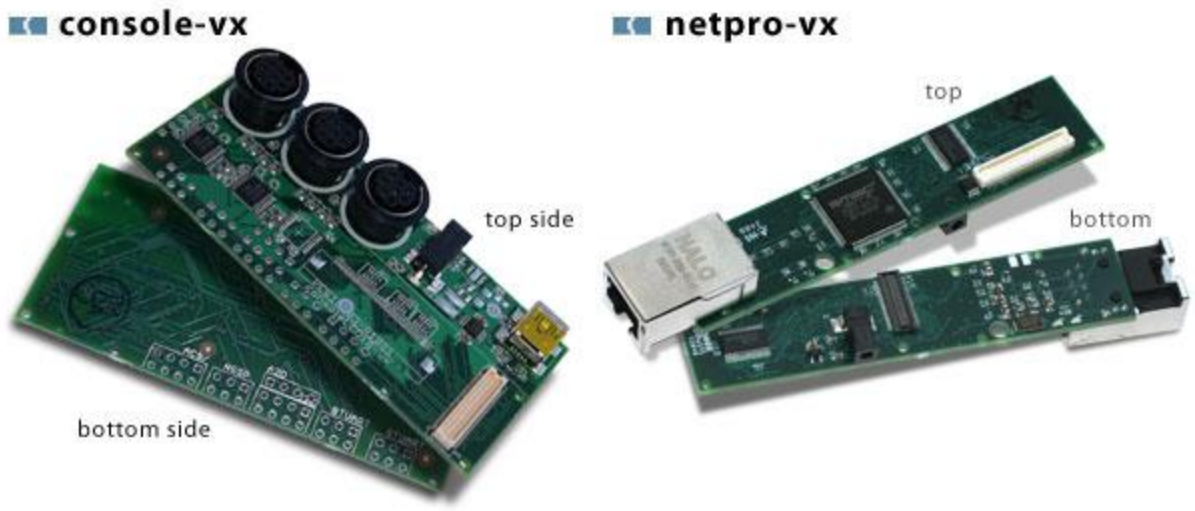

Figure 5.1: Console-VX and Netpro-VX Expansion Boards [32,33]

The Console-VX, shown on the left in Figure 5.1, is modified by cutting down the serial port headers and remains attached when the platform is added to the plane. The Console-VX connects to the bottom of the Gumstix itself and the NetPro-VX connects to the top. This makes the NetPro much easier to add or remove to the hardware stack. The Netpro-VX is only utilized when updating software or testing on the bench. This is mostly due to an increase in the profile of the device and a steep increase in power usage when the Netpro-VX is sending and receiving data. There are also additional expansion boards available for adding microcontrollers and other IO to the platform. As stated before, much of the decision was made due to the Gumstix's small size. Figure 5.2 shows the diminutive size of the Gumstix hardware stack. After modification of the Console board's serial headers, the thickness is about 0.278" which easily fits within the 1.23" thick wing.

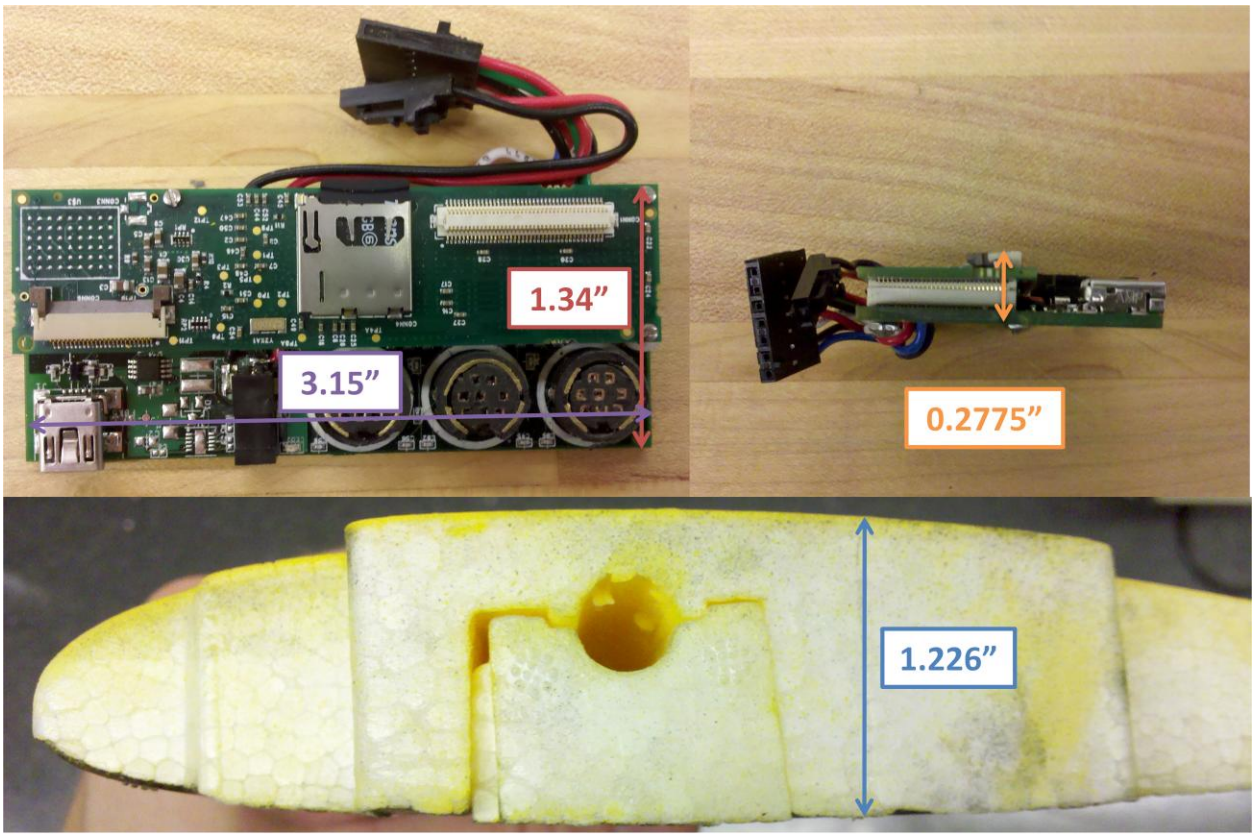

Figure 5.2: Dimensions of Gumstix and Wing

As well, the Gumstix is a device that has been used in the lab on many other projects, so it is a familiar platform where the build tools and environment are available and understood. Current consumption is approximately 270-285mA with the Console-VX attached and the system under load, and the Gumstix can utilize the same 5V source the FCS currently uses. The cost of the units is also low, at $169 for the COM, $25 for the Console-VX board, and $60 for the Netpro-VX board.

5.2 MCS Physical Layout

The MCS is positioned between the XBee PRO modem and the MiniFCS. There is a header on the MiniFCS that usually contains jumpers to connect TX, RX, and ground pins of the

XBee module that is attached beneath the MiniFCS to the FCS microcontroller. In this scenario, the MCS intercepts the data with cables that go to the serial ports on the Gumstix console board. Figure 5.3 illustrates the design. The Gumstix has a serial cable that connects to the FCS microcontroller UART output and another that connects to the XBee module. So while the XBee is still physically mounted to the FCS, all traffic is controlled by the MCS.

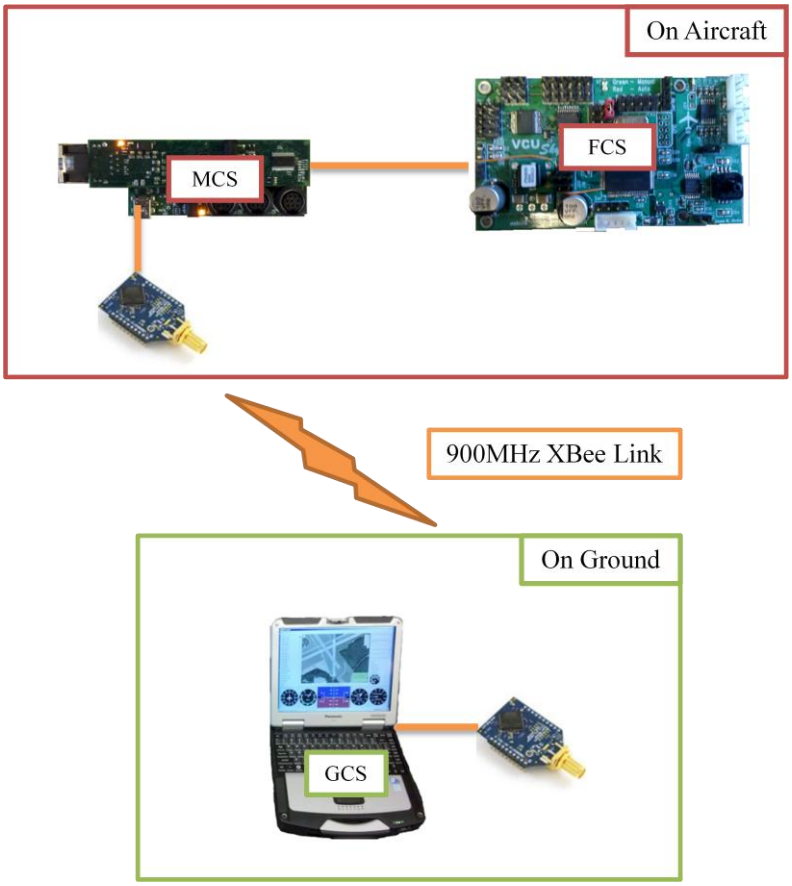

Figure 5.3: System Hardware Layout

The Gumstix Console-VX board contains IO for three serial ports [34]. They are a Full-Function UART, a Bluetooth UART, and a Standard UART. The FFUART was used as the console in Linux, so it was not used to maintain the ability to use the serial port as a console in case of emergency. The BTUART is the API port, which is used to read incoming data from the

GCS as well as send data to the GCS. The STUART is used to communicate to and from the FCS in the VACS protocol via the microcontroller UART. As was determined in the latter stages of MiniFCS development, the XBee modem needs to utilize hardware handshaking to maintain functionality. This resulted in a small wire being connected from the CTS pin on the XBee to a UART pin on the FCS. Now that the Gumstix is controlling the XBee this cable had to be moved to the CTS pin on the Gumstix BTUART that controls the XBee. The BTUART and the FFUART both have hardware flow-control, but since the FFUART is used as the console, the BTUART was the only option for API connectivity. The STUART maintains a 57,600 baud link as was initially setup in the FCS code, and the BTUART is connected to the XBee at 115,200 baud.

Due to the compact nature of the test vehicle, there is little room for electronic equipment and even less once the FCS and other devices are added to give it autonomy. This led to the placement of the MCS inside one of the wings. Figure 5.4 shows the Gumstix secured with Velcro on the bottom side of the left wing. This location is uniform on all gliders. All wiring is routed through the wing spar channels and into the fuselage to connect to the FCS and power source.

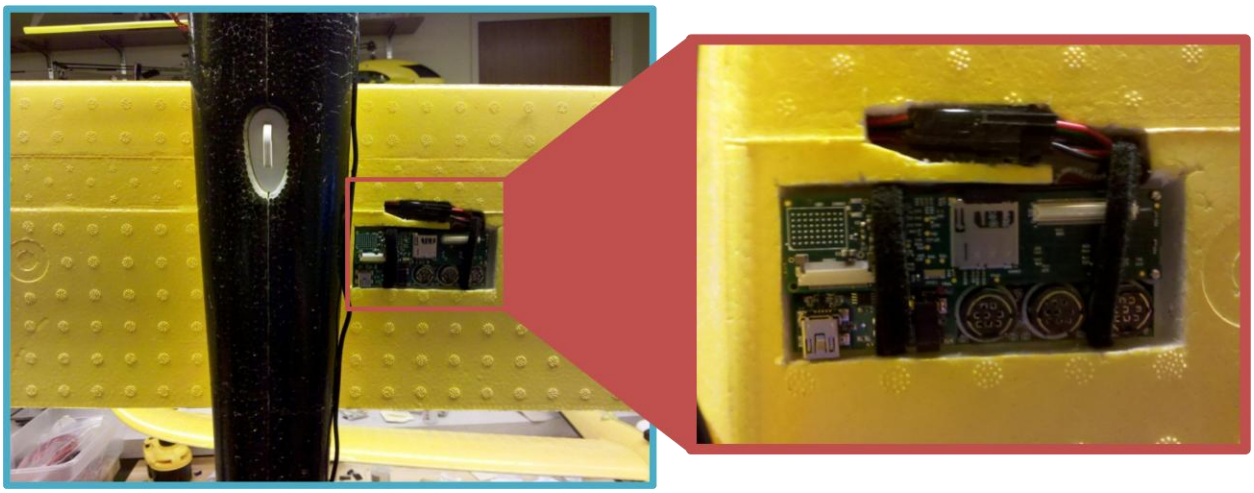

Figure 5.4: Physical Placement of MCS

The weight is kept as close to the fuselage as possible to have a minimal effect on flight performance. Connectors are used to make the MCS easy to remove or allow added freedom to mount the Netpro-VX board while still attached to the plane. This makes software updates and even simulation capable while the MCS is still within the plane.

Chapter 6: Mission Control System Software

6.1 MCS Main Loop

The MCS software has two main objectives: transmit and receive all serial data to and from the FCS and GCS and facilitate any collaborative operation. The current iteration of the MCS software operates at 200Hz which means an iteration time of 5 milliseconds. Figure 6.1 shows the startup screen which displays relevant data about the vehicle and settings. Figure 6.2 describes the main loop of the MCS code, which is started via a shell-script on boot [35].

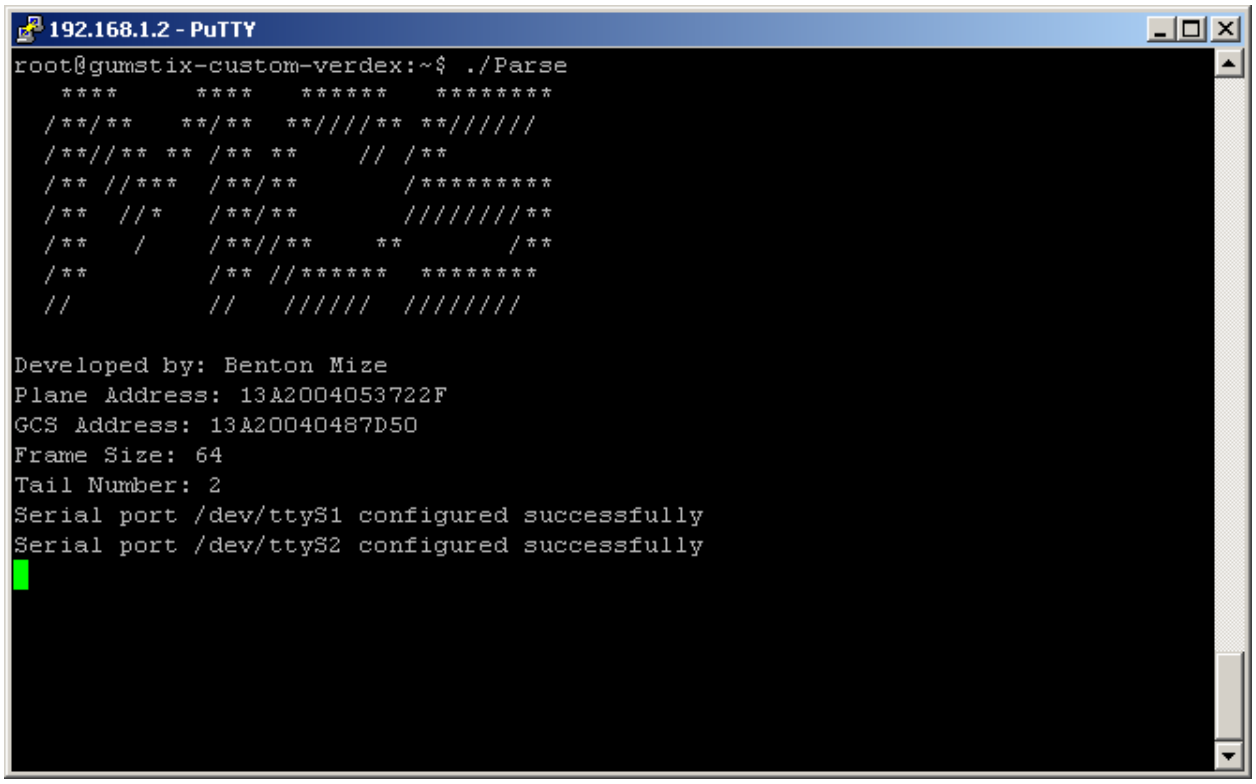

Figure 6.1: MCS Splash Screen

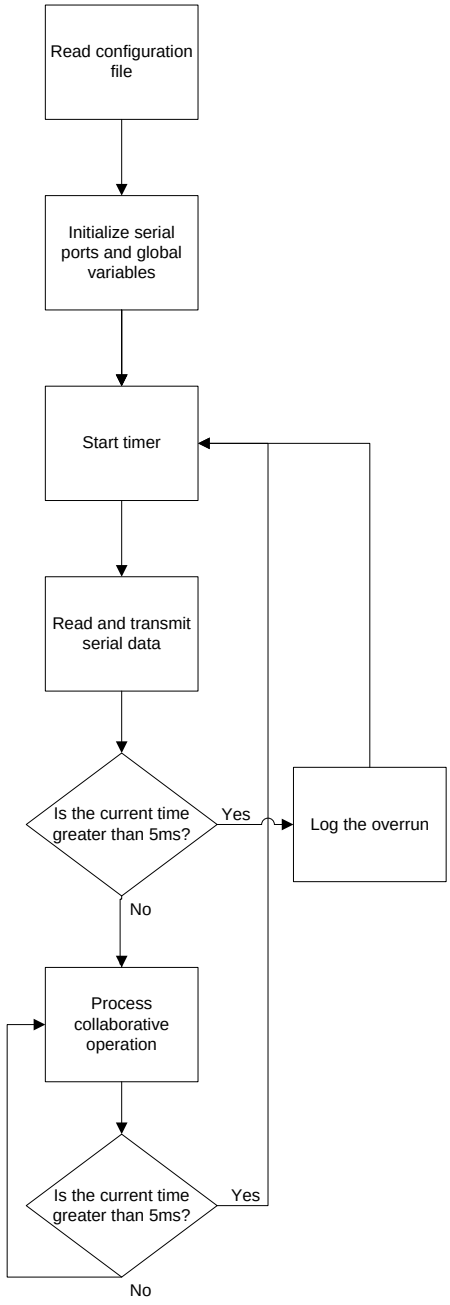

Figure 6.2: Main loop of MCS code

The MCS must transmit and receive serial data at all times. It is only when there is time left in the iteration that collaborative processing is done. This makes the distinction where the MCS is a soft, real-time system because collaborative operations do not, in this context, have deadlines that would lead to failures of the system. Each plane will initially read a configuration

file that contains information such as the tail number, planes modem address, GCS modem address, and frame size. This allows for quick changes of modems if necessary and easy editing of plane specific information.

Serial processing does conversions from the VACS format that is output by the FCS to the API format that is utilized by the GCS and XBee modems. Using this method the FCS does not need to undergo software changes to be included in the multiple UAV system. In addition to converting VACS data to API, the MCS also has a filtering system capable of gathering data that is in transit from the FCS to the GCS or vice-versa. This allows the MCS to gather information about the current state of the FCS such as speed, heading, altitude, navigation mode, etc. The additional data gives the MCS more control over the vehicle and allows it to make changes that were previously only able to be made by the GCS operator.

6.2 Collaborative Modes

Once serial data is processed, the loop will fall through to a case statement of collaborative operations. As shown below in Figure 6.3, currently there are only two states: "None" and "Search". The idea is that this can be modified to encompass new collaborative operations in the future. There are two local storage variables that hold the state information for the top-level state, or mode, and the sub-state. The "None" mode just wastes cycles until the iteration time is fulfilled. The "Search" mode calls the functions that process the search area command. Figure 6.3 shows the layout of the state machine with the sub-states that are within the top-level "Search" mode. When the top-level mode is set to "None" the "Search" sub-state is reset to prevent the system from entering the "Search" mode in an incorrect state at a later time.

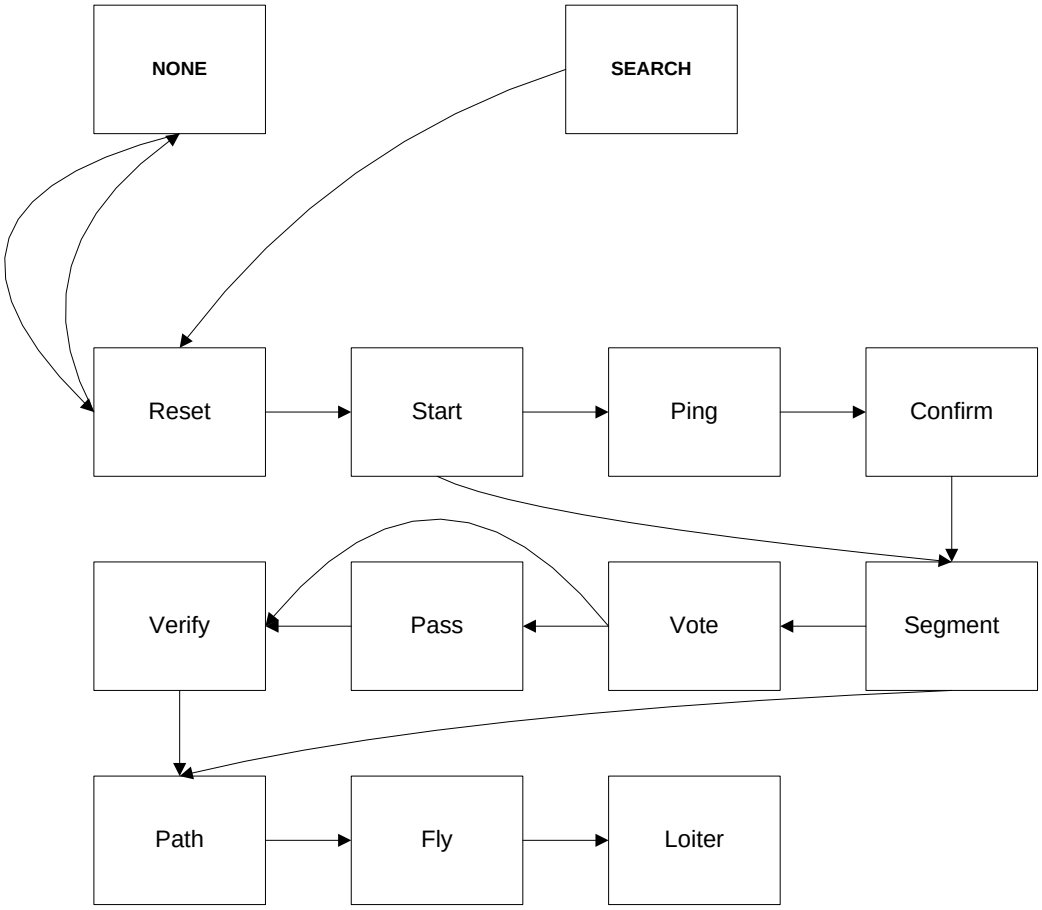

Figure 6.3: State diagram for collaborative modes

6.3 Search Area FSM

As discussed in chapter 4, the search area operation is initiated by a command that specifies the search region. As a brief summary of the operation, it starts by resetting and initializing some variables. Then the planes ping each other and determine the number of planes they can communicate with. Next the planes segment the designated search area and vote on their desired segment. Finally, the planes plan their search path, fly the determined pattern, and return to a loiter circle around their start position. The following sections will discuss this process in greater detail.

6.3.1 Reset

Once the search command is received by the plane the sub-state is changed to "Reset" and, it stores the points of the rectangle locally, as well as the number of planes the GCS reports it is aware of. It then requests the current navigation settings from the FCS. This has to be done because the MCS will need to make changes to settings such as navigation mode, but still needs to know the other settings (airspeed, altitude, etc.) so that it does not overwrite them with incorrect values. This state is also responsible for resetting the list of planes that the vehicle stores locally. This includes zeroing out any data like acknowledgements, address, and tail number that exist from previous iterations. Finally, the state is changed to "Start".

6.3.2 Start

Under the "Start" state, the system waits until the MCS has received an acknowledgement from the FCS that it has successfully gathered the navigation settings requested previously. Next, the MCS stores the current position of the aircraft, or the estimated center of the loiter circle if the plane is in loiter mode. The MCS gathers the altitude and battery voltage of the vehicle for use in later stages of the algorithm. It then builds a ping packet containing its own tail number, altitude, and battery voltage, in preparation for the next state. Then the system checks to see if the GCS reported finding more than one plane. If there is only one plane, then that plane is itself, and there is no need to ping for other planes or vote on segments. Therefore, if there is only one plane according to the GCS the next state is "Segment"; otherwise the next state is "Ping". This is signified in Figure 6.3 by the arrow skipping from "Start" to "Segment". However, the discussion will continue as though there are multiple aircraft as that is focus of this work.

6.3.3 Ping

The “Ping” state utilizes a timer process, which is utilized in many other states, where a packet is sent out periodically until a specified timeout is passed. In this case, the ping packet that was generated by the previous state is sent out at the rate defined by the user. The timing values for interval and timeout of the timer process are all able to be changed in real-time via a command form on the GCS. The MCS continues pinging until the timeout value is passed, and the state is changed to “Confirm”.

6.3.4 Confirm

The “Confirm” state is used to ensure that all planes have pinged the same number of planes as they were informed existed. If the numbers do not match the system goes into a pause mode, and notifies the GCS operator of the ping discontinuity. When the GCS receives this message it sends a pause command to all the vehicles. This particular search algorithm is globally convergent. Therefore, if the planes do not agree on the number of planes that exist then voting and segmentation in the algorithm will not converge, hence the discontinuity. If the plane numbers match, then the system transitions to the “Segment” state.

6.3.5 Segment

“Segment” also contains a special case if the number of planes is only one. In this scenario, the plane skips the voting and verification process, and goes directly to path planning mode. Otherwise, the area segmentation function is called. This function takes the area gathered from the initial search area command and segments it by the number of planes. It begins by gathering the distance for the length and width of the rectangle. Then the bearings are calculated

that represent the directions along the length and width. As shown in Figure 6.4, the length and left bearing are always referenced from coordinates zero to three, and the width and downward bearing are referenced from coordinates zero to one. Therefore, to change the directions and bearings the user must draw or rotate the image differently to achieve the desired effect.

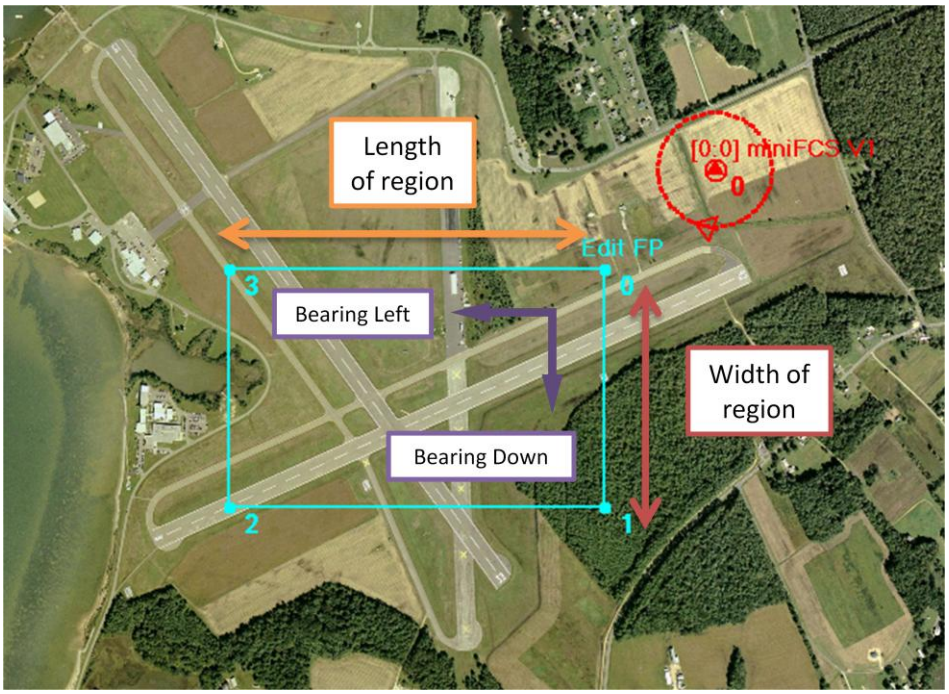

Figure 6.4: Dimensions and Bearings for Path Planning

Next, a segment width is calculated by dividing the length of the rectangle by the number of vehicles. A local array of points is used to store the coordinates for the corners of the segments to be defined. For the initial and final segments one side (two points) is already know because the perimeter of the rectangle is known. Next the system simply moves the next set of boundary points based on the segment width multiplied by an index until the number of segments is equal to the number of planes. The centroid of the segment is calculated along with the distance from the previously determined start position of the vehicle to the centroid.

After the segments are defined the MCS sorts the segments in ascending order based on the distance to the centroid of each region. The list of known vehicles is also sorted in ascending order based on battery voltage with a fallback sort based on the tail number. In other words, in the event of identical battery voltages the system will give the plane with lower tail number a higher priority. At this point, all the planes will contain the same list of planes in the same order. The distinction is that may have organized their lists of segments differently because their locations, and thus their distances to the centroids, will be different. At the end of the "Segment" state the plane with the highest priority creates a token and defines a voting packet with its segment choice. Now the state is changed to "Vote".

6.3.6 Vote and Pass

When in the "Vote" state, only the plane with the token is able to communicate. This means the highest priority vehicle, ideally the lowest battery voltage, gets to vote first and thus gets its highest priority segment. This state also utilizes the timer process previously discussed to send out its vote to other planes. In this scenario as well, if the voting timeout is elapsed a discontinuity is reported and all planes stop until the user takes action. When a vote is sent to another plane, that plane removes the segment from its list of available segments and changes its vote if necessary. Then the receiving plane sends a voting acknowledgement to tell the plane with the token that the vote has been received and its segments amended accordingly. When the plane with the token receives an acknowledgment from all vehicles it transitions to the "Pass" state, where it uses the timer process to send the token to the next plane in the list. Again, if it is unable to pass the token, operation is paused, and the operator informed. When the token is successfully passed, the receiving plane sends an acknowledgement of reception and the

transmitting plane moves to the “Verify” state, where it waits until the voting process is complete.

6.3.7 Verify

The token passing process continues in the “Vote” state until the last plane receives the token. Because of the uniform priority of the vehicles, the last plane is aware that it is the final vehicle and destroys the token. It then moves to the “Verify” state, where the other planes should be waiting for the process to complete. The “Verify” state only contains functionality for the last vehicle. When the last vehicle is in this state it sends a message to each plane telling them that the voting process is complete. This causes all vehicles to transition to the “Path” state.

6.3.8 Path

The “Path” state, as its name describes, is where the planes take their regions and construct a path for searching the area. This was a development that went through a great deal of simulation to determine the turn and associated layouts to minimize the out-of-region flying. In this design, it is acceptable for the plane to operate outside of the designated search area to facilitate turning and lining up on the rhumb lines.

The two most important aspects of the search area are the altitude of the aircraft and the field of view of the image sensor. The altitude is gathered using the filtering technique in the “Start” state, and the FOV is currently an arbitrary angle in degrees assuming a fixed downward facing camera [36]. Using simple trigonometry, the width of the sweep can be determined by doubling the opposite side of the two right triangles from the reference of the FOV on the

aircraft.

$$sweepWidth = 2 * \left(altitude * \tan(\frac{1}{2} * FOV)\right)$$

This does make the assumption then that the roll angle of the aircraft will be zero, hence the need to operate outside of the boundaries for turning. Figure 6.4 illustrates the camera position and calculation for sweep width.

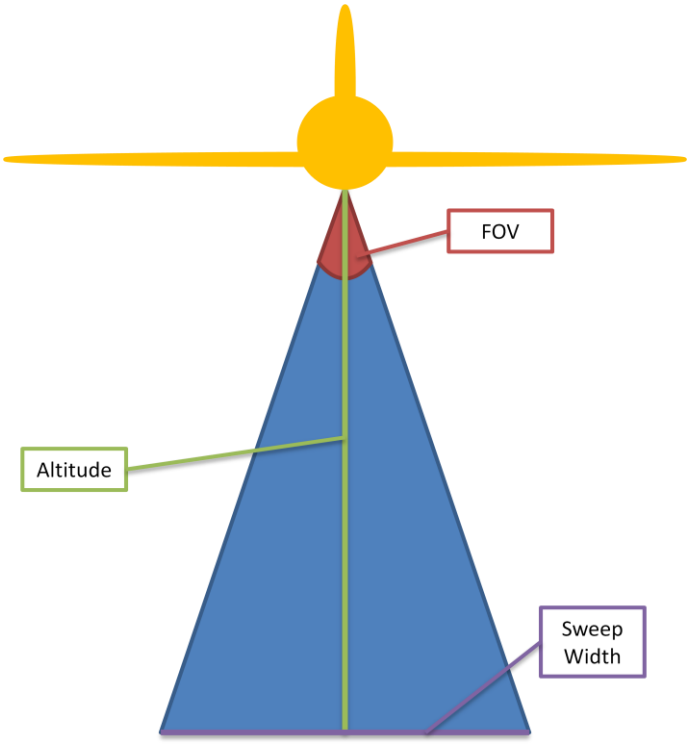

Figure 6.5: Sweep Width Calculation

The sweep width is the main measurement needed because it determines the viewable area of the aircraft. Using the known width of the segment to be searched, the number of sweeps can be determined before the path is actually planned. This is done by simply dividing the length of the area to be searched by the sweep width. This value is stored and used to index the loop iterations later in the process.

There are a variety of parameters that can be changed to manipulate the turn. Turn distance defines the length of the leg that extends from the search area to allow the vehicle to make an 180˚ turn. Various simulations yielded a turn style shown in Figure 6.6. The figure is divided into a top and bottom portion, where the bottom represents the sweep legs that would be in the search area. The top is the portion where the vehicle is operating outside of the search area to turn onto the next sweep. Parameters shown include "Ratio", so named because it is set as a ratio of the turn distance used. For instance, a ratio of 1.0 is 100% of the turn distance yielding a squared turn pattern and, as shown in the figure, a lower ratio such as 50% would yield a more knife-like pattern. The "Minimum Turn Distance" is usually set as the sweep width, but the user can set a minimum turn distance or set the turn distance to a specific value to control the distance given for the UAV to turn on to the next leg.

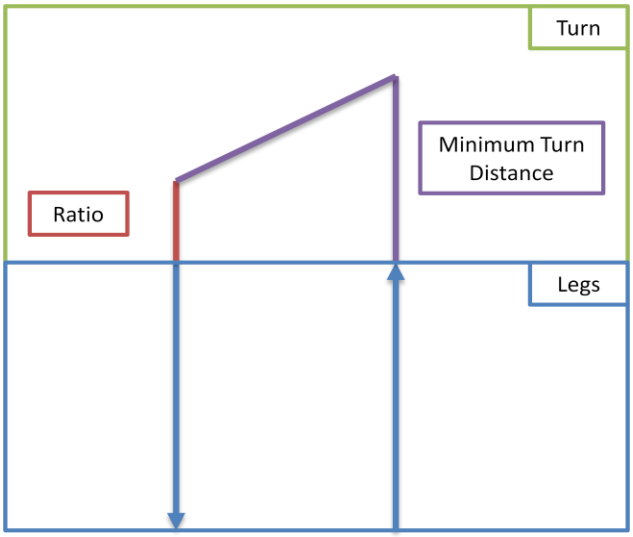

Figure 6.6: Turn Style

There is also an overlap parameter, which will cause the sweep width to be reduced by the value set. This is used to compensate for variation in roll and give the plane some room for error. An

enter distance parameter was added to change the initial length of the leg built from planes initial position to the search pattern. This is separate from the turn distance value to allow more control over the planes entry to the operation.

The actual path planning begins by using half of the determined sweep width and moving the initial corner of the region inward and then moving it up based on the enter distance parameter. This motion is described in Figure 6.7, where the first point is created based on the enter distance parameter set by the operator. The system will begin with a downward sweep. A variable is alternated to facilitate the back-and-forth sweeping motion desired.

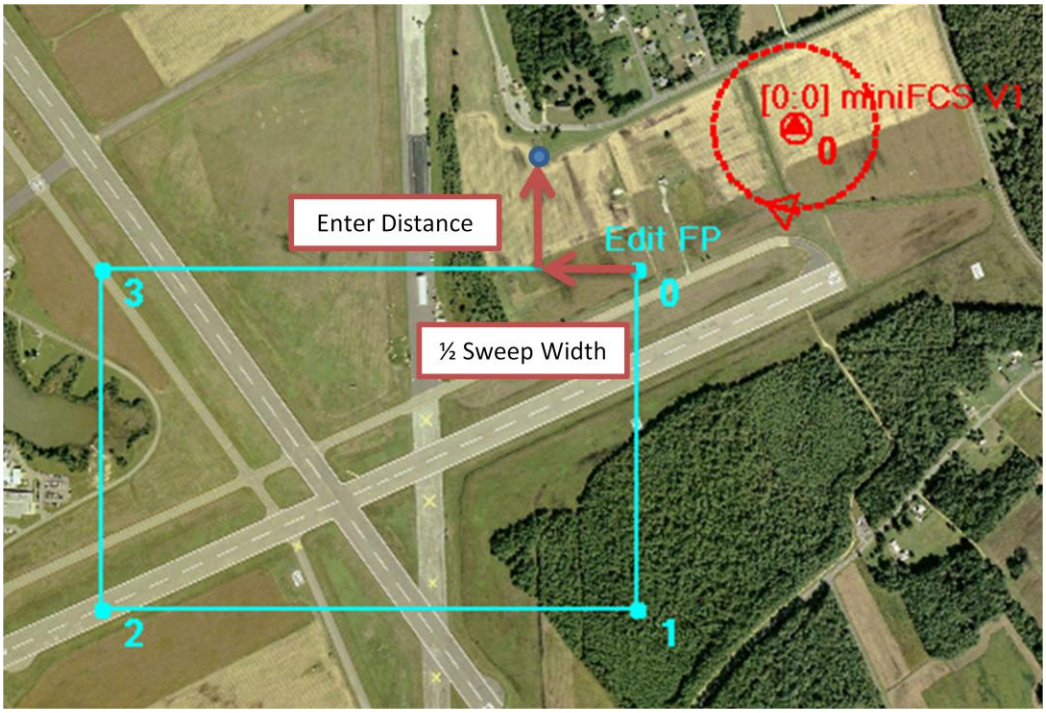

Figure 6.7: Enter Distance Layout

Next, as Figure 6.8 describes, the direction is turned downward and the point is moved back down by the enter distance value, to put it back on the perimeter of the search area. Then, the point is moved down again by the sum of the segment's width plus the arrival range of the

aircraft. Each waypoint has an arrival range that defines the radius of a circle around each waypoint where the aircraft considers that point to have been reached. Adding the arrival range here means that the vehicle will not consider the point to have been attained until it is out of the search area. Each movement is based on the latitude and longitude position of the previous point. The path is created by moving a single point around and storing its location as a waypoint in the total path. This method is used so that it is simpler to change the layout of the pattern.

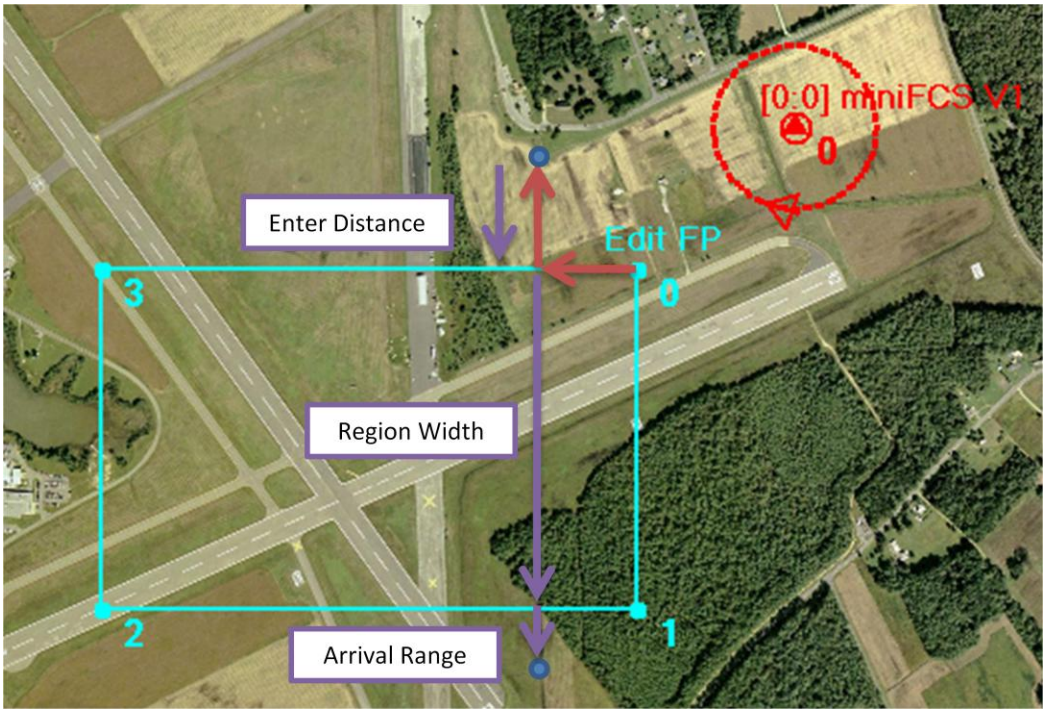

Figure 6.8: Sweep Layout

The next three waypoints are the construction of the turn itself. The waypoint continues to move downward based on the turn distance, then shifts left by twice the sweep width. Then the point is shifted up by the ratio multiplied by the turn distance, where a waypoint is added then the point is moved up by the remaining ratio (1-ratio) multiplied by the turn distance, which

places the point back on the perimeter of the region. Figure 6.9 shows the construction of the turn using the parameters discussed.

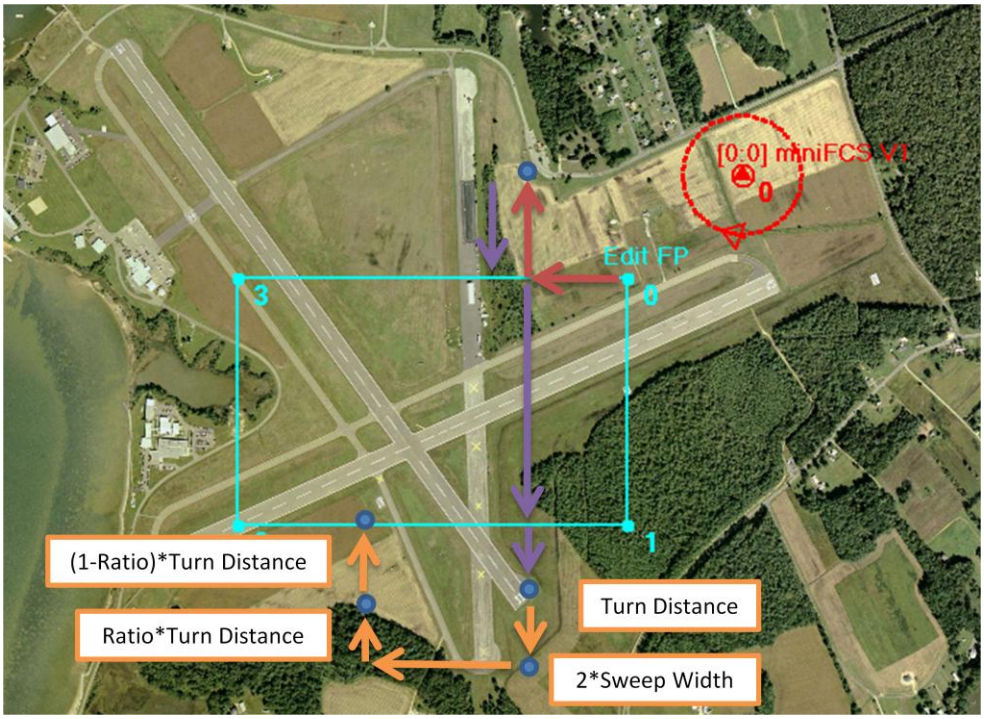

Figure 6.9: Turn Construction

The last turn point is kept on the edge of the region because it aides in the transition of cross-track lead distances that change from turns between legs of the region. The longer the lead, the longer and smoother the transition back into the rhumb line. However, the longer lead distance also means the plane will take longer to adjust back onto the rhumb line, and lead distances less than about 30 meters typically cause the system to oscillate over the rhumb line which is a highly undesirable trait. As stated before, there are parameters in place that allow the operator to change the lead distances for turns and legs when the system is in flight. The process of creating the sweep leg and turn continues for a preset number of iterations. When this process is finished the last waypoint is set to the starting waypoint collected in the "Start" state.

Originally, the path planning process would add only the width of one sweep and continue up and down until the left-most sweep was complete, then move across the pattern to the start position much like the raster pattern seen in [15]. This is still an effective method, as shown in Figure 6.10, but has some drawbacks when the areas are particularly small.

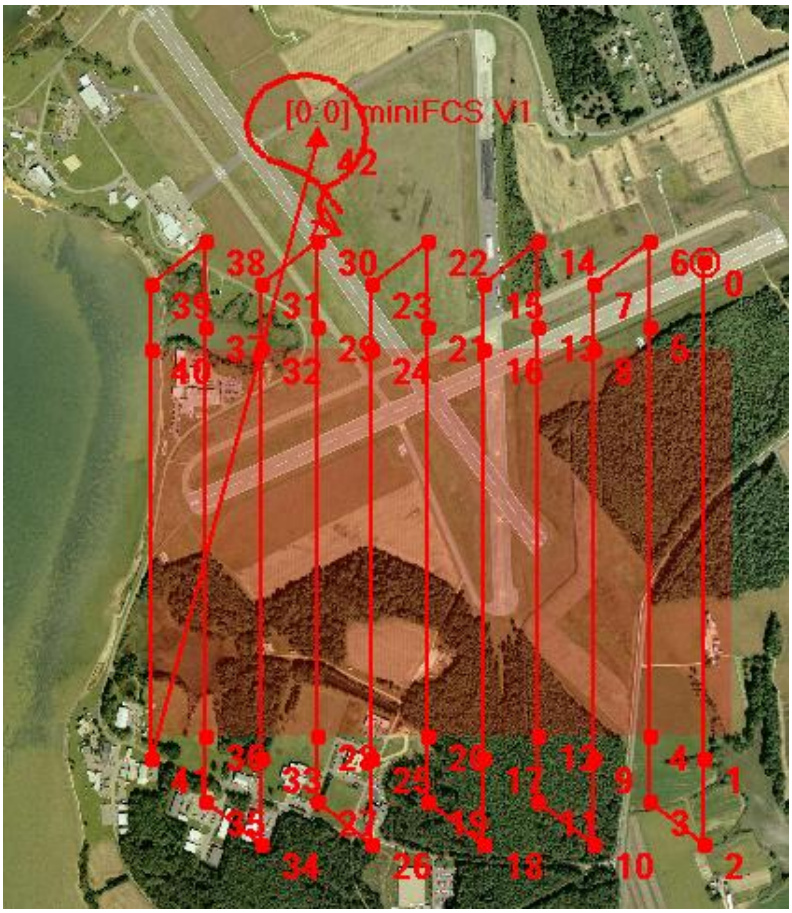

Figure 6.10: Simple Lawnmower

This was recognized when the time came to do field testing in smaller areas, whereas simulations had been done over large regions and the issues were unseen. In very small areas it is difficult to get a dense sweep pattern for viable testing. Parameters can be set to force an output with more sweeps, but this results in shorter distances between sweeps. This presents a problem where the plane must go farther and farther away to make its turns or risk high roll

angles when entering the search area. This was solved by using an interlacing scheme, where the plane will skip a sweep, doubling the width it has to turn onto the next leg of the search pattern. Then when the plane has reached the edge of the region it turns back and sweeps the legs that were skipped. This is the reason for shifting the waypoint by twice the sweep width. It also has the added benefit of exiting closer to where the plane began, instead of crossing the entire pattern to return to start. Figure 6.11 illustrates a complete interlaced pattern by a single plane (the plane can be seen as the red triangle moving toward loiter point 26). As shown, there is always going to be a point where the plane must turn within a single sweep width which occurs when the vehicle must transition back across the search area.

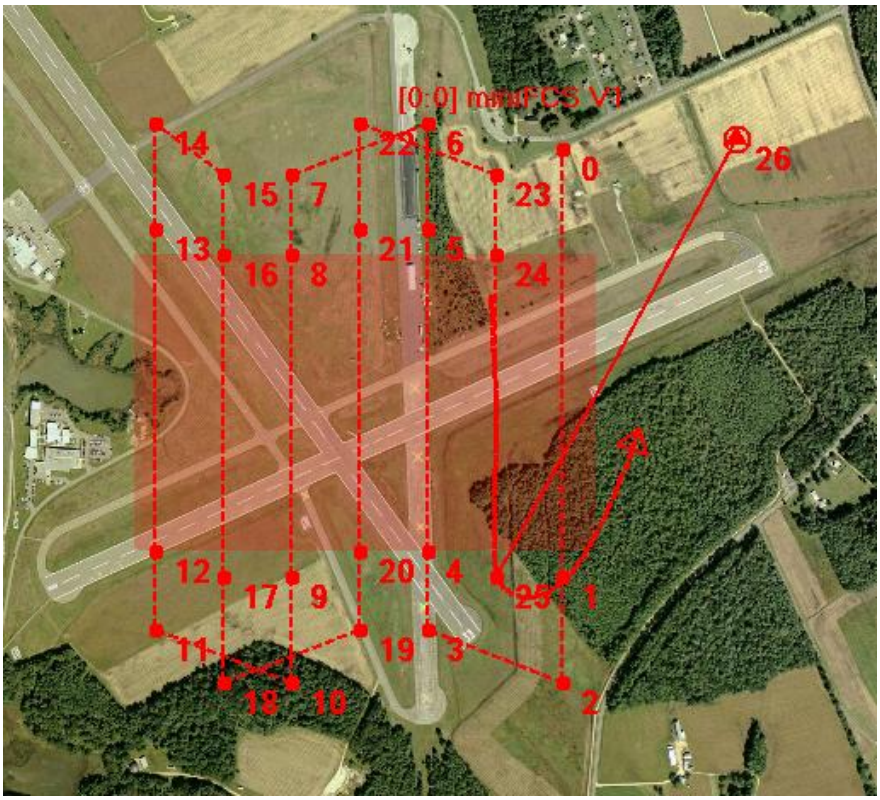

Figure 6.11: Interlaced Lawnmower

Once the path planning stage is complete the system contains an array of the waypoints needed to search the area. These are kept in local storage onboard the MCS. The vehicle then transitions to the "Fly" state.

6.3.9 Fly and Loiter

Because of restrictions on the VACS payload and the large size of waypoints, it is difficult to reliably send more than about 19 waypoints to the FCS at a time. For the search mechanism, this limit would easily be surpassed since there are five waypoints per turn, so the plane only receives data in subsets (as mentioned in chapter 4). This works by using the filtering system to determine the current waypoint the plane is approaching. When the final waypoint for that subset has been attained the MCS sends the next set of waypoints to the FCS for the next leg of the pattern. Therefore, the "Fly" state is essentially monitoring the current waypoint value of the FCS and sending the next set of waypoints at the appropriate time. This eliminates the limitation on the length of waypoints that are able to be sent to the FCS.

The "Fly" state is also responsible for changing the cross-track lead distance depending on whether the vehicle is in a turn or sweep leg. It is advantageous for the lead to be larger in the sweep legs and smaller in the turns. There was also a change made to take the FCS navigation mode out of cross-track and back into simple attractor mode [7] at the end of the turn distance to the "ratio" waypoint in each turn. Figure 6.12 illustrates the locations of the navigation mode changes to facilitate a rounder turn and thus smoother approach back to the search area.

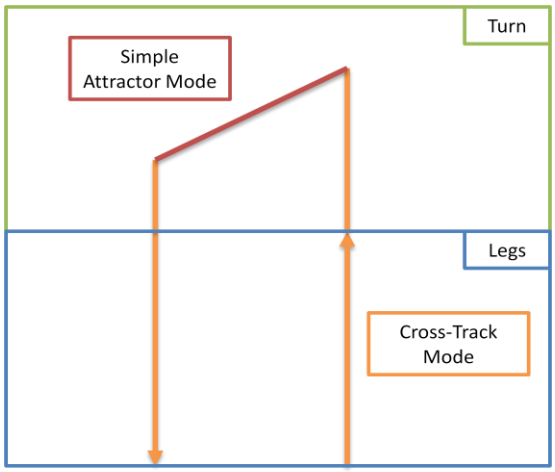

Figure 6.12: Navigation Modes in Turns

In addition, the MCS periodically pings to see what vehicles are still in range of itself; this is purely a test feature to get an idea of the range of the vehicles themselves. The MCS also checks to see if the vehicle is approaching the final waypoint. In this case, the MCS will change the flight mode of the vehicle to a clockwise loiter pattern with radius of 50 meters. This final point, as stated before, is also the starting point that was initially determined in the "Start" state. This state is now changed to "Loiter", the final state of the search area system. The actual "Loiter" state is just an empty state where the vehicle stays as it awaits the next command from the operator.

At any time during the search area operation, if the operator re-sends the search the vehicles will cancel the current operation and restart from their current position. As well, if the operator enters a standard waypoint pattern via the user interface the MCS will see the command going to the FCS and stop the collaborative operation.

6.4 MCS Status Messaging

The MCS sends a status packet at 2Hz that contains data about the state, mode, and current flight path of the collaborative system. Figure 6.2 shows the data that is stored in the status packet.

Table 6.1: Status Packet Structure

Status Packet					
Current FCS Set	Total Sets	Current Waypoint	Mode	State	Updates

Each field in the table is only a single byte, generating a total of six bytes. As mentioned before, the GCS does not receive all waypoints for a search path at one time, so fields in this packet aid in the construction of the multiple sets on the ground. The current FCS set refers to the current set of waypoints that the FCS is operating on. The "total sets" field refers to the total number of sets of waypoints that make up the entire search pattern. The current waypoint is the waypoint that the FCS is on with reference to the entire pattern. In other words, it is the FCS set value multiplied by the current waypoint in the current set to give the operator a waypoint ID within the set of all points in the search path. The mode is an enumerated operating mode where the current options are "None" and "Search". The state is the state of operation in the mode, which was shown in the Figure 6.3 describing possible states and modes. The "Updates" field is used to signify when data is ready for the GCS or to notify the GCS of errors in the collaborative operation. Table 6.2 shows the layout of the "Updates" byte.

Table 6.2: "Updates" Bit-Level Layout

Updates							
Unused	Unused	Unused	Verify Discontinuity	Pass Discontinuity	Vote Discontinuity	Ping Discontinuity	Search Path Ready
7	6	5	4	3	2	1	0

Bits seven through five are unused and available for expansion. Bits four down to one are the method utilized to signify a discontinuity to the GCS operator and trigger the context sensitive button on the UI.

The least significant bit position is used for the "Search Path Ready" flag. This signifies when the path planning operation has been completed and the stored list of waypoints is ready to be sent to the GCS operator. Even though the entire list is not sent at a single time to the FCS, the GCS operator needs to see the entire path to be sure the vehicle is operating correctly. When the flag is high the GCS sends a request for the first set of waypoints to the MCS. The MCS then sends down the first set and the GCS continues to request the points until the number of sets requested equals the number of total sets for the entire pattern. Then, the entire pattern is displayed for the GCS operator. This system allows the GCS to take control of the messaging system, and use the system of retries and acknowledgements that already exists to facilitate the transfer.

Chapter 7: Results and Flight Testing

Much of the testing was done in simulation using a combination of software and hardware simulation. Initially, much of the testing of the algorithm and state machine could be done simply using the MCS and FCS, without any feedback or modeling of actual flight. Ultimately the system had to be fully simulated, including an accurate aircraft model, and this was done using a version of an in-house developed hardware-in-the-loop interface board. The HILS board was updated to a newer Xilinx Virtex-4 FPGA platform and replicated to create a complete set for use for hardware simulation of all vehicles. The complete HILS-board simulation setup, shown in Figure 7.1 below, uses the open-source flight simulation software FlightGear [12].

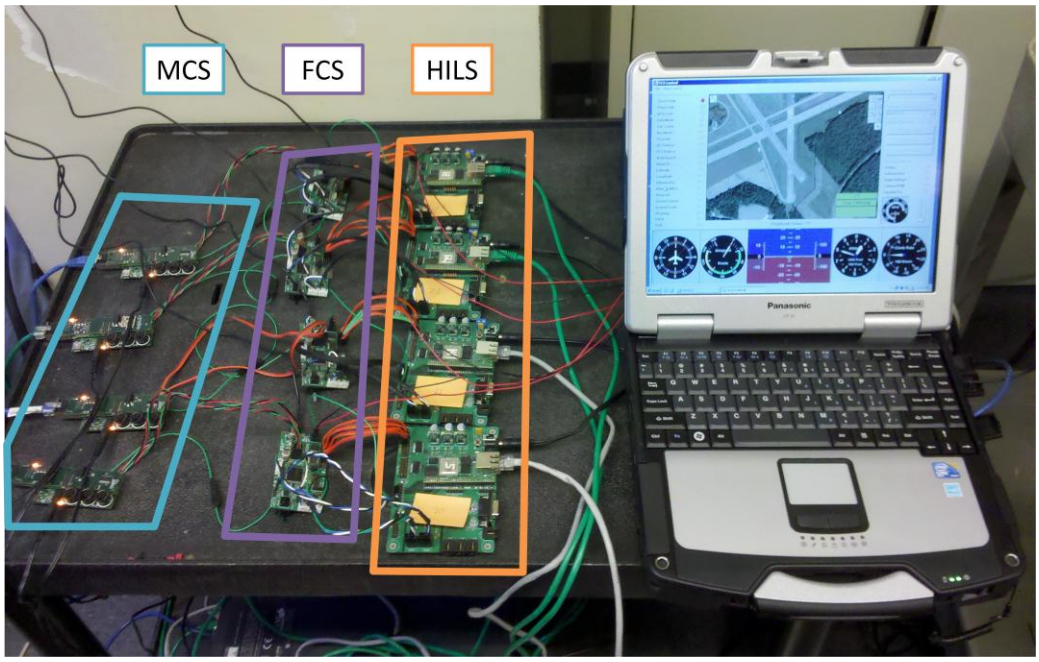

Figure 7.1: Initial HILS Setup

As shown, this setup enabled hardware simulation of up to four vehicles with visual feedback via the GCS. This capability is pivotal to test the functionality of the collaborative operation, in that it allows the testing of communication network to go further, and to more realistically test flight operations with the identical hardware that will be added to the physical vehicles. A simple program was also developed to convert the saved API formatted log file on the GCS into a VACS log file. This enables the use of previously generated extraction and graphing utilities to be reused.

This bench top setup has also been modified to a rack design that allows simulation without removing the electronics from the vehicles. It also allows the servos on the vehicles to be plugged into the HILS and give visual confirmation of correct flight surface movement. Figure 7.2 shows this new setup.

Figure 7.2: Rack Style HILS Setup

Ultimately, a bench top setup, as shown previously in figure 7.1, will exist in addition to the rack setup to provide a quicker development and test system.

7.1 Collaborative Vote Validation

For the testing of the collaborative system on the bench, the code is modified to allow the user to input a battery voltage. This needs to be done to simulate the vehicles having different priorities and test the voting process to ensure the vehicle with lowest battery voltage gets the closest segment. The user is able to input a voltage level on the command line that will signify the battery voltage for the test.

Table 7.1: Plane Description Chart for Voting Test

Tail	Color	Voltage (V)
2	Red	10V
3	Blue	11V
4	Yellow	12V

The voltages given, tail number, and color, corresponding to the images from the GCS shown in previous figures, and in the figures below are shown in the table 7.1. In this test, all vehicles start from the same initial loiter point and are given a search area command. Ultimately the solution segment sequence, from right to left, should be red, blue, and yellow respectively. Figure 7.3 shows a zoomed in view of the vehicles in their loiter pattern before receiving the search command. Each vehicle is signified by a triangle with a series of dots trailing behind representing previous GPS locations.

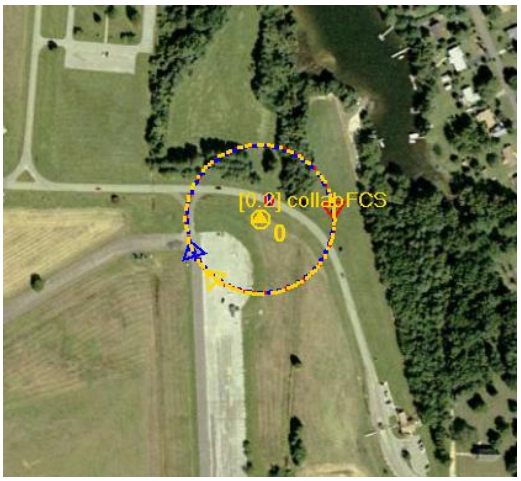

Figure 7.3: Initial Loiter Location

The search area is directly below the loiter point and expands left (West). Figure 7.4 shows that the correct choices were made by each vehicle, where vehicle two (red) chose the closes segment, vehicle three (blue) the center, and vehicle four (yellow) the most distant segment.

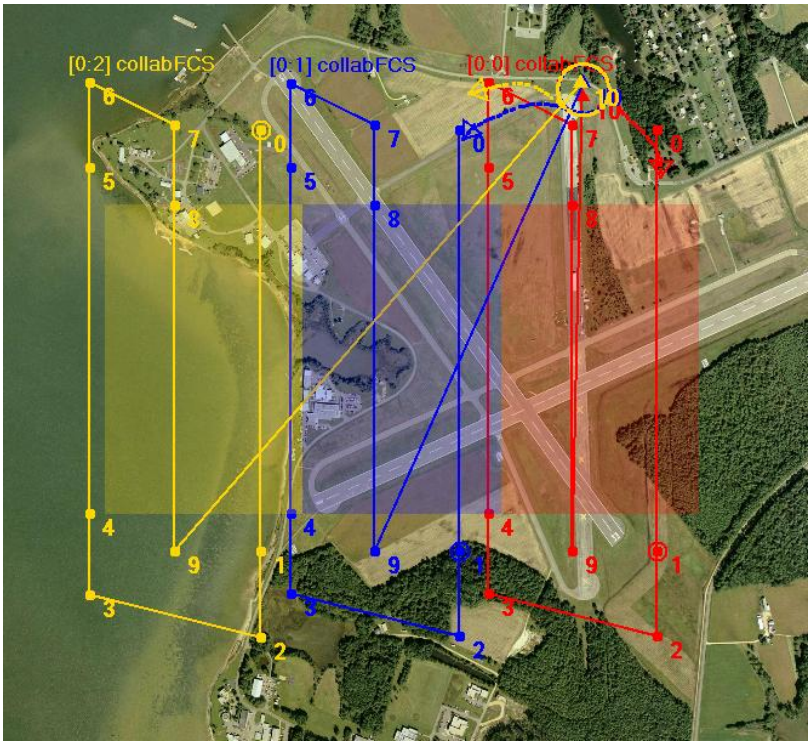

Figure 7.4: Display of Voting Output

7.2 Speed Comparison

To compare the speedup in applying the same pattern to four planes versus one the following test was conducted. A single plane was given a pattern to fly and then a set of four planes was given the same pattern and flight paths and times were recorded. Figure 7.5 shows the path traveled by the single vehicle and the desired waypoint path. Waypoints are numbered (from zero to 34) to show the progression of travel.

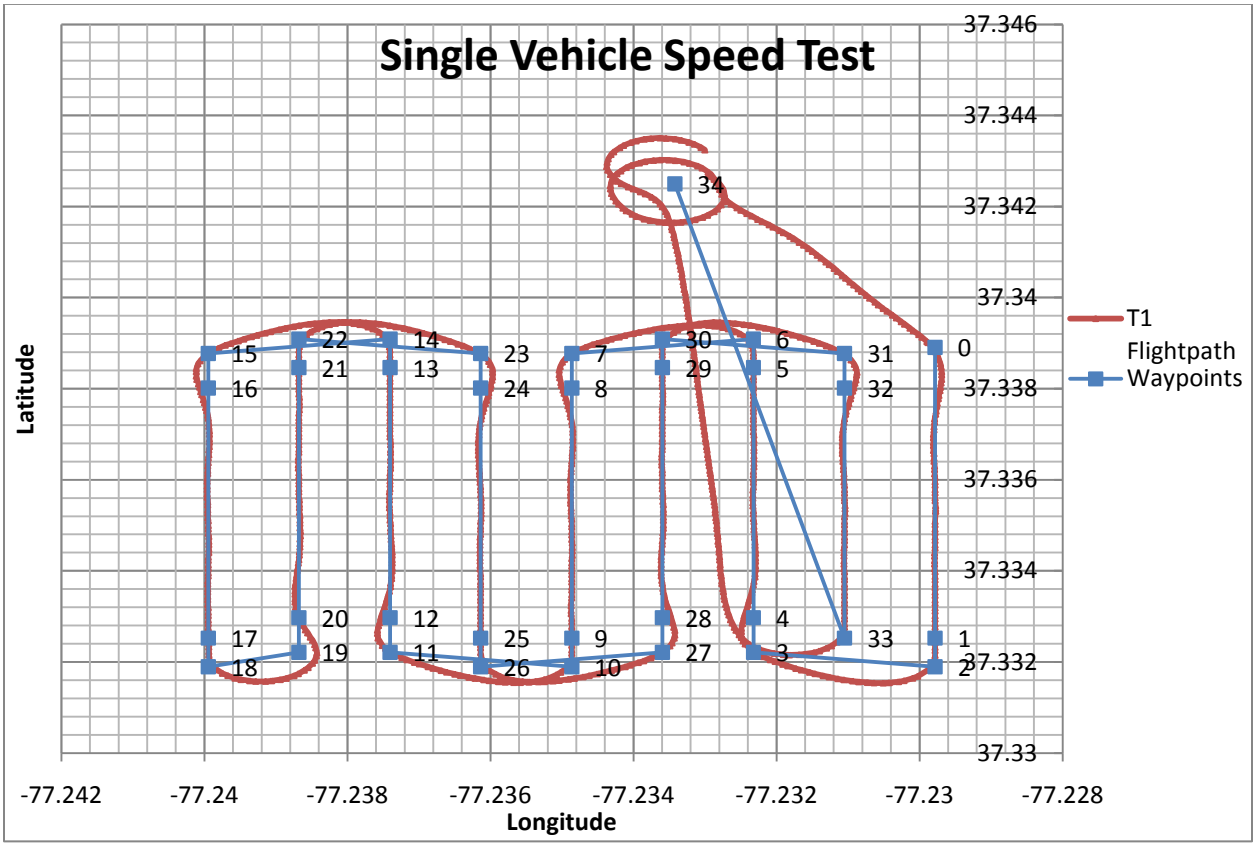

Figure 7.5: T1 Base Speed Test Simulation

Next, the same test was run where each vehicle was sent the search command. The vehicles determined the appropriate segment based on the priority generated and proceeded to plan paths for each subsection. Results are shown in Figure 7.6.

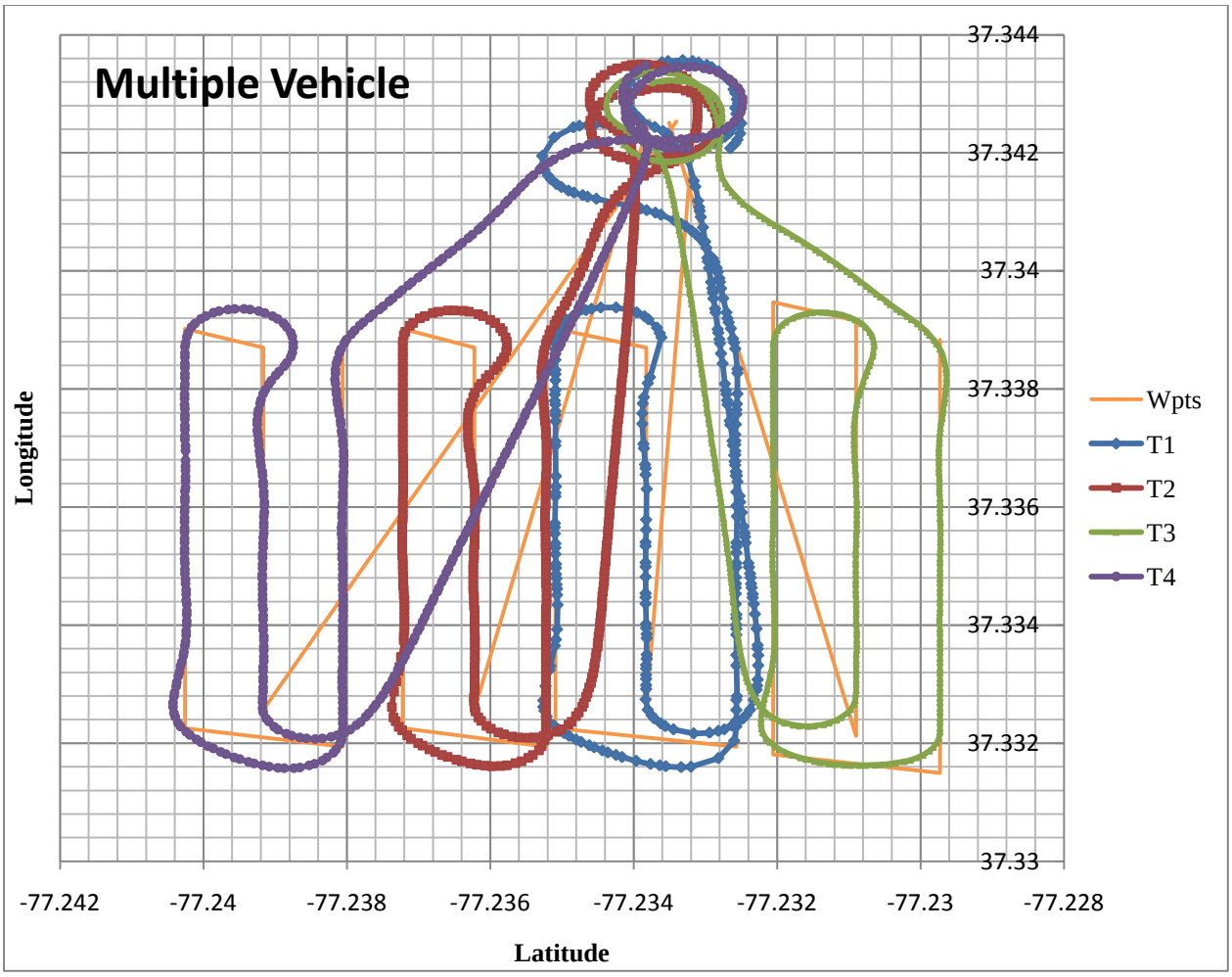

Figure 7.6: Four Vehicle Speed Test Simulation

This test gives some advantage to the single plane in that the vehicles operating altitude is 170 meters and when the other vehicles are tasked each one operates at that altitude or lower. This is to replicate a similar scenario in actual test flight where collision avoidance is achieved by operating at different altitudes. Lower altitude will yield longer patterns as the sweep width will be reduced, so the additional vehicles are not given individual advantage over the single vehicle. All other parameters between the vehicles are identical. The individual aircraft altitudes are displayed in Table 7.2.

Table 7.2: Operation Altitudes in Speed Test

Tail	Altitude (m)
T1	170
T2	140
T3	160
T4	150

The fight times were recorded from the initial display of the flight path on the operator screen to the first turn into the loiter pattern as the vehicle completes the pattern. These are shown below in Table 7.3.

Table 7.3: Time Comparison for Speed Test

Planes	Start	Finish	Elapsed
1	04:05.8	13:13.5	09:07.7
4	18:48.8	23:00.5	04:11.7

Overall, the four planes were able to complete the task at 45.956% of the time it took the single vehicle. Majority of time lost using the multiple vehicles is spent moving to the start position of the pattern and returning from the pattern to the start position. Unlike the multiple planes, the single plane definitely spends the majority of time in the search pattern and less in transit.

7.3 Wind Simulations

Wind is a constant issue for smaller aircraft and that certainly includes the vehicle used in this experiment. Because of the variations found in the everyday environment, as well as the importance of effective cross-track in a search operation, the vehicles were simulated to analyze the effect of wind on the current FCS in the search mode. Simulations were done using wind from the East to the West at speeds of 5, 10, and 15 knots. The results of these tests are shown

below in Figures 7.7 – 7.9. Waypoint patterns are shown with the flight path overlaid. Each vehicle enters to the right of its waypoint set and exits back to its loiter position.

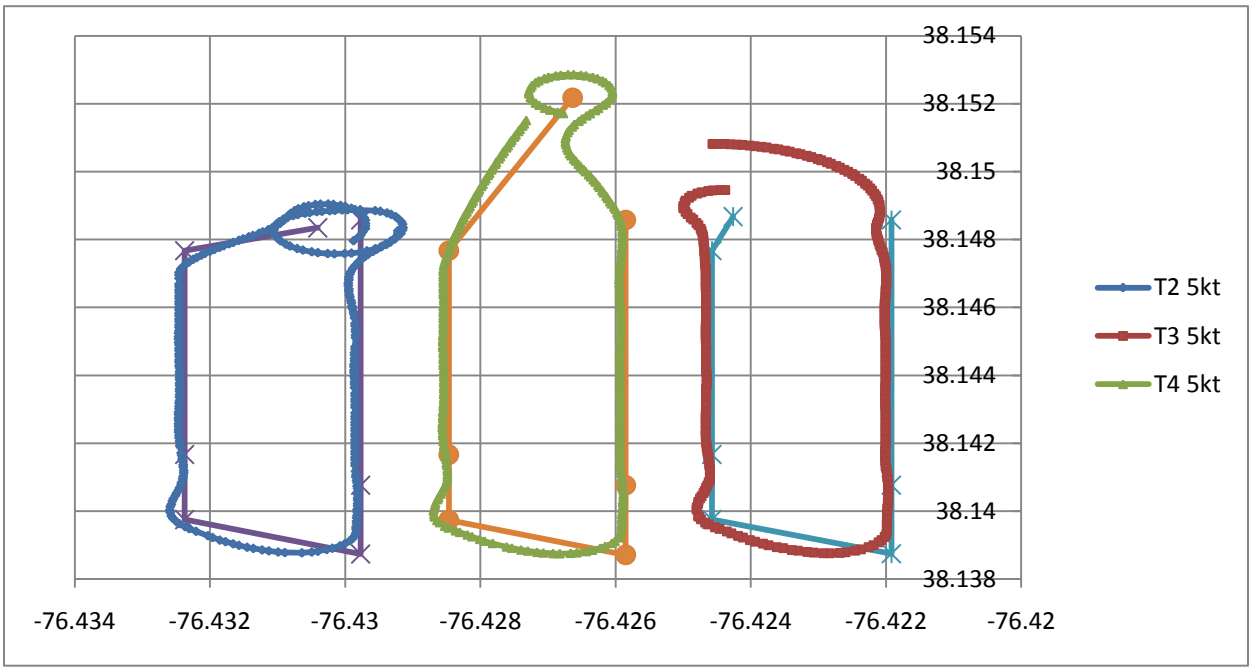

Figure 7.7: 5 Knot wind test

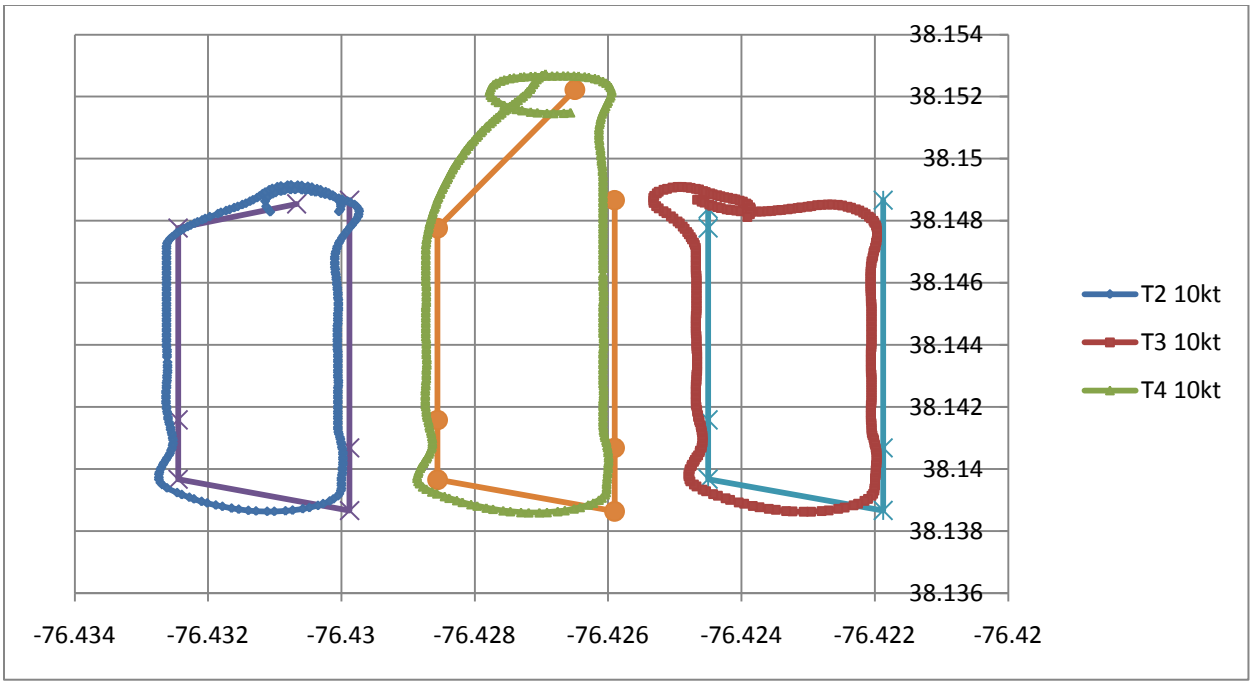

Figure 7.8: 10 Knot wind test

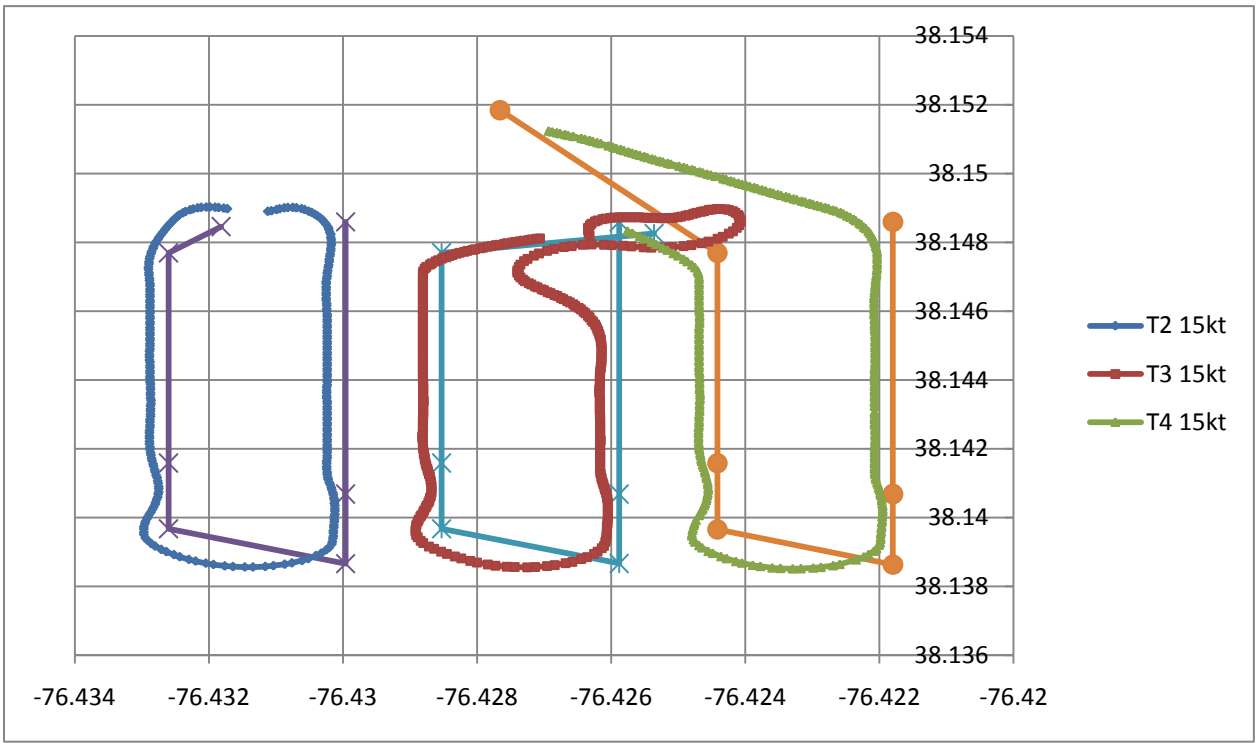

Figure 7.9: 15 Knot wind test

Wind is definitely an issue and it seems to stem from the cross-track system internal to the FCS. The vehicle goes into a constant yaw, towards the source of wind, and begins to fly at an offset of the desired pattern. This is especially prevalent in the 15 knot test. Offsets from the rhumb line are approximately 7m, 15m, and 24m, from 5 to 15 knots respectively. The vehicle can utilize a shorter cross-track lead in an effort to pull the vehicle closer to the rhumb line more frequently but this has limitations, as discussed before, where a cross-track lead that is too short will cause oscillations over the rhumb line. The lower speed wind test issues can be overcome by adding some overlap into the system. As stated before, this is a parameter that already exists and can be used to compensate for these effects. In higher winds, such as those in the 15 knot test, the cross-track system will need to be modified to account for the accumulated error that is more problematic in a system that relies on close following of the rhumb line.

7.4 Flight Testing

Flight testing was done with a group of two vehicles, specifically tail numbers one and four. These vehicles utilized the same search parameters and airspeeds. Different altitudes were used to avoid collisions when searching and traversing to waypoints. T1 operated at 180m while T4 operated at 200m. Each vehicle had its own safety pilot and two ground operators were used to view the data for each plane and coordinate with the safety pilots in case of emergency.

Figure 7.10: Initial Positions and Designated Area

To actually test the search area, the system utilized a smaller FOV for the camera to generate a useable pattern. At the altitudes the vehicles were operating, a more realistic FOV would have yielded patterns consisting of a single sweep, so for the sake of testing this value was reduced to generate a larger pattern. Figure 7.10 shows the actual area tested and the initial positions of the aircraft. These positions will be the starting points for the vehicles in the remainder of the figures.

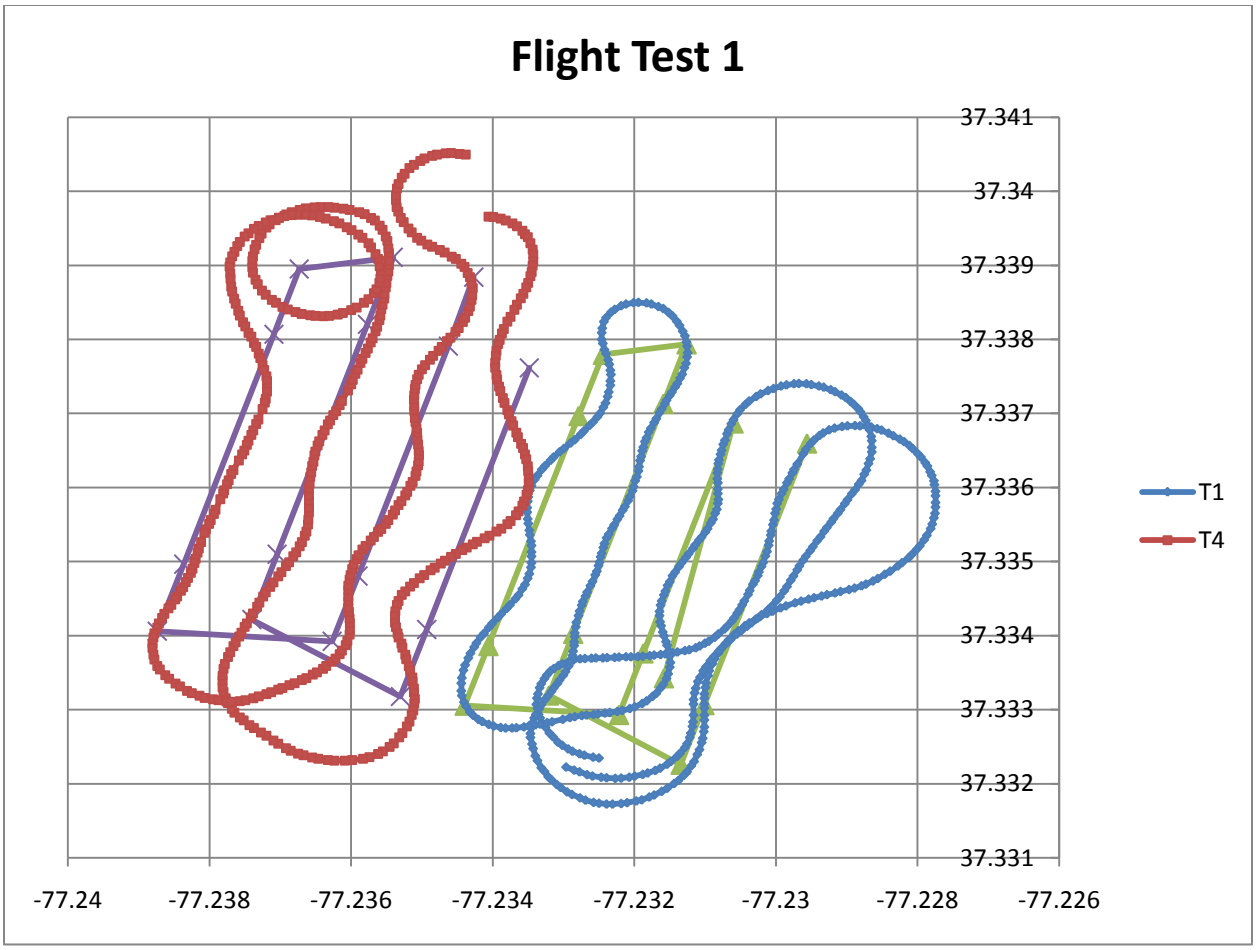

Figure 7.11: First Flight

The results of the first flight test are shown above in Figure 7.11. Tail number four had an issue on the seventh waypoint in the first flight, as shown in Figure 7.10 (the circled waypoint). Apparently the arrival range was set too low and it was unable to turn enough to gather the waypoint. The vehicle was given a command to move to the next waypoint by the operator and the arrival range was increased for the next test. There are definite oscillations on the rhumb lines among both vehicles that cannot be directly attributed to wind. However, the vehicles successfully collaborated to complete the pattern and return to their initial loiter patterns.

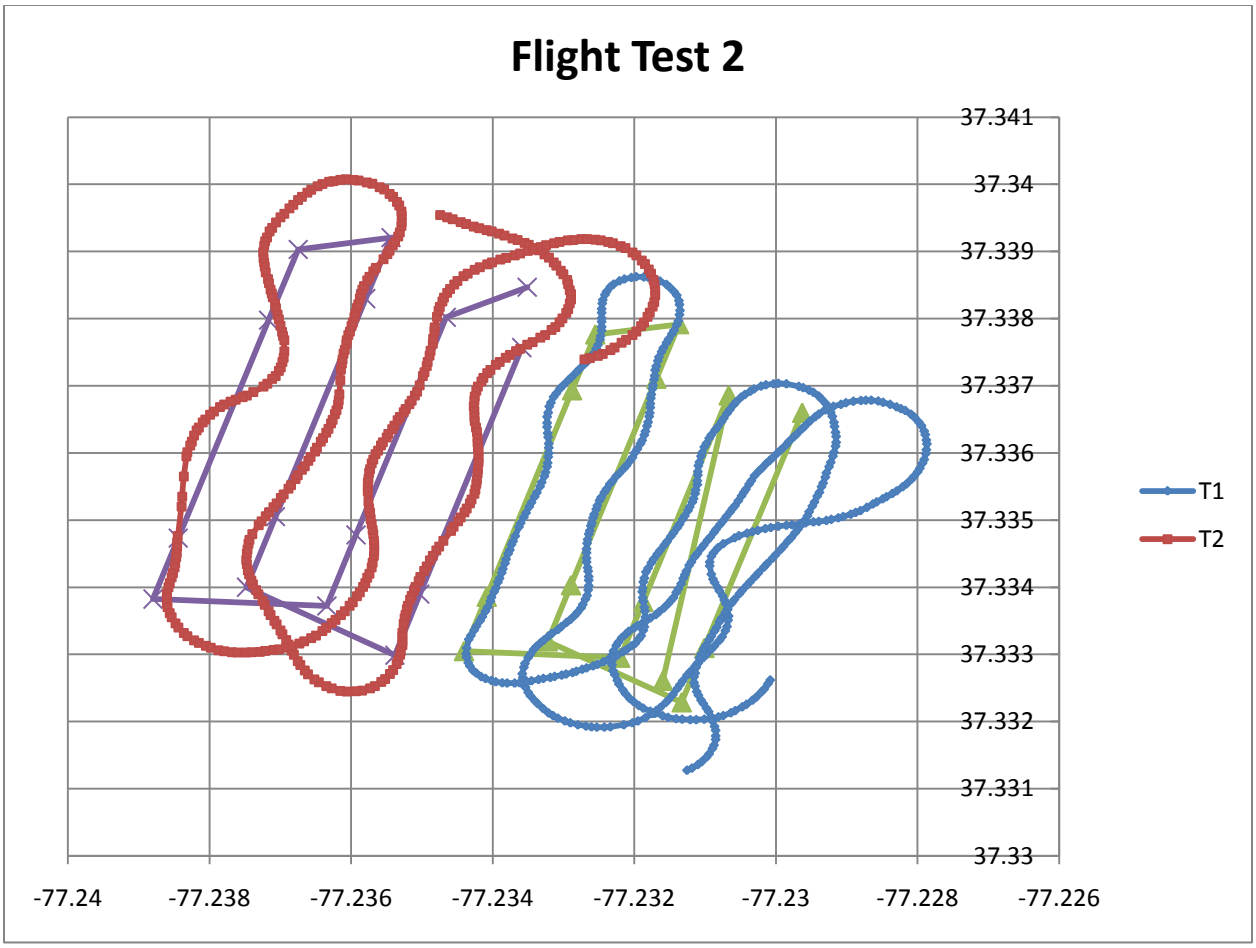

Figure 7.12: Second Flight

Flight two is a similar depiction of flight one, only without the circling due of waypoints. Cross-track ability definitely needs improvement, but much of this error can be alleviated with additional tuning of the vehicles. Even still, the vehicles are distinctly following the search path, and the utilization of the simple attractor mode in the turns seems to have benefitted the transition from turn to search area. Vehicles did correctly select their segments in the voting process as is shown since each vehicle got the segment closest to them. As well, given the range between vehicles and the smaller scale vehicle antennas, there were no communication problems amongst the vehicles when coordinating the search area operation. All in all, the vehicles

successfully segmented, voted, planned, and traversed the area, given the collaborative information formed amongst them.

Chapter 8: Conclusions and Future Work

8.1 Conclusions

The goal of developing a collaborative system for multiple aircraft was successfully achieved. As well, the system was not only hardware simulated, but also physically flight tested using the previously designed MiniFCS platform. This testing and development has allowed the use of this system for future development of collaborative UAV operations at VCU.

The MCS was developed on the Gumstix platform and can continue to be used to develop more features and collaborative modes. The segmentation of the MCS from the FCS gives flexibility in the future for changes of FCS platform as well as more advanced collaborative algorithms independent of the capabilities of the FCS. The MCS allows greater control over the FCS and enables more complicated operations to be done at a layer above the direct control of the vehicle and its sensors while limiting the amount of management done by the operator. MCS code has been developed to allow for future developments to be easily added to the current state machine and the Gumstix has plenty of processing power.

The GCS has been updated to allow multiple vehicles per controller and can be utilized in the future for larger groups of aircraft. The control interface has been changed to display collaborative specific data and give the user greater control over the participating vehicles. It has also been designed to allow for expansion of the API mode to other modems as well. Some arbitration mechanisms have been investigated but need further development for system viability.

The results show that a collaborative communication system has been designed and successfully simulated and flight tested. The choice of separation of the MCS from the FCS allows for expansion as well as flexibility. The GCS upgrades also now allow future research to control multiple vehicles with only a single modem and operator. The overall system operates well and is capable of continue use as collaborative algorithm research progresses.

8.2 Future Work

The clear future work would be the development of more advanced collaborative algorithms. New operations such as tracking, formation flight, or continuous surveillance could be tested. As well, the current search area method could be made more dynamic or less globally convergent.

There are many updates that could be made to this system. One potential update is to utilize a new radio, such as the Wi-Fi Bullet [37]. This system utilizes a 5.8GHz radio with built in Ethernet. Because the MCS has the capability to add the Netpro-VX board, which is currently not used in flight due to SWaP requirements, the addition of this radio should be relatively simple. The ability to utilize custom firmware in the Wi-Fi Bullet also would allow the development of a protocol tailored to our uses while reducing unnecessary overhead. Of course, a serious advantage would be the much larger bandwidth, up to 54Mbps. This sort of update would probably require a change of platform to something with more payload space and power storage. As well, with more bandwidth the parameters that describe the image sensor could be replaced with an actual camera.

Another aspect that needs to be addressed is communication arbitration. This has proven to be a difficult problem that is currently limiting greater expansion of the system. Adding bandwidth via a new modem is one solution, but probably not the most efficient. There are many possible schemes that could be used, but one way to make the utilization of these schemes easier would be to have a synchronous clock between all the vehicles. This could potentially be extracted from GPS or via an expansion of the Gumstix [38]. With a synchronous clock token passing and polling could be done without the limiting transmission overhead that was seen to diminish success in testing.

References

[1] Zak Sarris, "Survey of UAV Applications in Civil Markets," STN ATLAS-3 Sigma AE and Technical University of Crete, June 2001.

[2] B. Lorin. (2011) FDNN - Fire Department Network News. [Online]. http://www.fdnntv.com/Evergreen-Unmanned-Systems

[3] Unmanned Editor. (2001, May) Unmanned - Ground, Aerial, Sea and Space Systems - Oregon nurseries explore unmanned drone technology to monitor fields. [Online]. http://www.unmanned.co.uk/unmanned-vehicles-news/unmanned-aerial-vehicles-uav-news/oregon-nurseries-explore-unmanned-drone-technology-to-monitor-fields/

[4] C. Dillow. (2011, February) PopSci. [Online]. http://www.popsci.com/technology/article/2011-02/dod-wants-warfighting-robots-can-collaborate-battlefield

[5] A. Ryan, M. Zennaro, A. Howell, R. Sengupta, and J. K. Hedrick, "An Overview of Emerging Results in Cooperative UAV Control," *43rd IEEE Conference on Decision and Control*, vol. 607, p. 602, December 2004.

[6] (2011) UAV Collaborative - Projects. [Online]. www.uav-applications.org/projects.html

[7] J. E. Ortize, "Development of a Low Cost Autopilot System for Unmanned Aerial Vehicles," Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, Master's Thesis 2008.

[8] A. Ryan et al., "Decentralized Control of Unmanned Aerial Vehicle Collaborative Sensing Missions," *American Control Conference*, pp. 4672-4677, July 2007.

[9] Cloud Cap Technology - Flight Management Systems - Piccolo Systems. [Online]. http://www.cloudcaptech.com/piccolo_system.shtm

[10] A. Ryan et al., "A Modular Software Infrastructure for Distributed Control of Collaboration UAVs," *Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit*, August 2006.

[11] J. How, E. King, and Y. Kuwata, "Flight Demonstrations of Cooperative Control for UAV Teams," *AIAA 3rd "Unmanned Unlimited" Technical Conference, Workshop, and Exhibit*, September 2004.

[12] (2011, July) FlightGear. [Online]. www.flightgear.org

[13] R. W. Beard, T. W. McLain, D. B. Nelson, D. Kingston, and D. Johanson, "Decentralized Cooperative Aerial Surveillance Using Fixed-Wing Miniature UAVs," *Proceedings of the IEEE*, pp. 1306-1324, July 2006.

[14] R. Beard et al., "Autonomous Vehicle Technologies for Small Fixed Wing UAVs," *AIAA J. Aerospace, Comput., Information, Commun.*, vol. 2, pp. 92-108, January 2005.

[15] Henrik Grankvist, "Autopilot Design and Path Planning for a UAV," Defense and Security, Systems and Technology, FOI Defense Research Agency, Stockholm, 2006.

[16] Jarurat Ousingsawat, "UAV Path Planning for Maximum Coverage Surveillance of Area with Different Priorities," *20th Conference of Mechanical Engineering Network of Thailand*, October 2006.

[17] S. A. Bortoff, "Path Planning for UAVs," *Proceedings of the American Control Conference*, June 2000.

[18] L. Rognant, J. M. Chassery, S. Goze, and J. G. Planes, "The Delaunay contrained triangulation: the Delaunay stable algorithms," *1999 IEEE International Conference on Information Visualization*, pp. 147-152, July 1999.

[19] R. J. Szczerba, P. Galkowski, I. S. Glicktein, and N. Ternullo, "Robust Algorithm for Real-Time Route Planning," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 36, no. 3, pp. 869-878, July 2000.

[20] R. H. Klenke, "A UAV-Based Computer Engineering Capstone Senior Designe Project," *IEEE International Conference on Microelectronics Systems Education/International Symposium on Multimedia Software Engineering*, pp. 111-112, 2005.

[21] Q. Cheng, "The Development of an FPGA-Based Autopilot for Unmanned Aerial Vehicles," Electrical Engineering, Virginia Commonwealth University, Richmond, VA, Master's Thesis 2006.

[22] W. C. S. IV, "The Development of a Linux and FPGA Based Autopilot System for Unmanned Aerial Vehicles," Electrical Engineering, Virginia Commonwealth University, Richmond, Master's Thesis 2007.

[23] R. C. DeMott, L. B. Mize IV, and R. H. Klenke, "Control Algorithms Used in the VCU Rotor-Wing Flight Control System," *Proceedings of AIAA Infotech@Aerospace*, April 2010.

[24] R. C. DeMott II, "Development of a Flexible FPGA-Based Platform for Flight Control System Research," Electrical and Computer Engineering, Virginia Commonwealth University, Richmond, Master's Thesis 2010.

[25] Digi International Inc. (2010, June) Digi - XTend RF Module. [Online]. ftp1.digi.com/support/documentation/90000958_C.pdf

[26] (2011) Digi - Products - XBee-PRO 900 RF Module - Product Manual. [Online]. http://ftp1.digi.com/support/documentation/90002134_B.pdf

[27] J. Cooper, R. C. DeMott, and J. Ortiz, "Low Bandwidth Streaming Protocol for UAV Communications," Virginia Commonwealth University, Report 2008.

[28] (2011, April) Wikipedia - Hidden Node Problem. [Online]. http://en.wikipedia.org/wiki/Hidden_node_problem

[29] P. C. Ng, S. C. Liew, K. C. Sha, and W. T. To, "Experimental Study of Hidden-node Problem in IEEE802.11 Wireless Networks," *ACM SIGCOMM 2005*, 2005.

[30] (2011) Gumstix Developer Center - Hardware Design. [Online]. http://www.gumstix.org/hardware-design/verdex-pro-coms/79-design-and-production/191-gumstix-verdex-pro-feature-overview.html

[31] (2011) Gumstix Developer Center. [Online]. http://gumstix.org/hardware-design/overo-coms/75-overview-and-roadmap/102-gumstix-overo-connector-overview-a-design-information.html

[32] (2011) Gumstix - Products - Netpro-VX. [Online]. http://www.gumstix.com/store/product_info.php?products_id=207

[33] (2011) Gumstix - Products - Console-VX. [Online]. http://www.gumstix.com/store/product_info.php?products_id=185

[34] F. B. Cvalcanti, "Powering and Connecting the Gumstix Verdex," Electrical Engineering, University of Brasilia, UnB, Brazil, Technical Note 2009.

[35] Vivek G. Gite. (2002) Linux Shell Scripting Tutorial v1.05r3 A Beginner's Handbook. [Online]. http://www.freeos.com/guides/lsst/index.html

[36] (2011, July) Wikipedia - Angle of View. [Online]. http://en.wikipedia.org/wiki/Angle_of_view

[37] (2011) Ubiquiti Networks - Bullet. [Online]. ubnt.com/bullet

[38] M. Meier, "Precise Time Synchronization for Wireless Sensor Networks using the Global Positioning System," Distributed Computing Group Computer Engineering and Networks Laboratory, Zurich, Thesis 2010.